THM | CAMPUS GIESSEN | MNI
TECHNISCHE HOCHSCHULE MITTELHESSEN
Mathematik, Naturwissenschaften und Informatik

Bachelorthesis

Plattformübergreifende Anwendung zur Darstellung von Quantenschaltkreisen mit Proof of Concept für Phase Kickback

zur Erlangung des akademischen Grades

Bachelor of Science (B. Sc.)

vorgelegt dem
Fachbereich Mathematik, Naturwissenschaften und Informatik
der Technischen Hochschule Mittelhessen
von

Cedric Schacht

20. Juni 2024

Referentin: Prof. Dr. Bettina Just
Korreferent: Prof. Dr. Steffen Vaupel

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Gießen, 20. Juni 2024

Zusammenfassung

Ein QuBit hat die Form $\alpha |0\rangle + \beta |1\rangle$. Wenn α und β beide ungleich 0 sind, spricht man von einem überlagerten Zustand oder von Superposition. Mehrere QuBits lassen sich als Register betrachten, kann ein Register nicht mehr durch Ausklammern in einzelne QuBits zerlegt werden, ist das Register verschränkt.

Ein Schaltkreis besteht aus einem Register auf dem Quantengatter angewendet werden. Jedes Quantengatter kann als eine Matrix dargestellt werden, welche die Auswirkungen des Quantengatters auf das Register beschreibt. Ein Messvorgang ist kein Quantengatter, das Messen eines überlagerten Zustands hat mit Wahrscheinlichkeit des Quadrats der Amplitude das jeweilige Ergebnis.

Registerzustände aus wenigen QuBits lassen sich als Würfeldiagramm visualisieren. Die Auswirkungen von Quantengattern und Messung lassen sich daran darstellen.

Ein Simulator der einen Quantencomputer widerspiegelt, braucht für die Berechnung komplexe Zahlen, sowie einige grundlegende Funktionen der linearen Algebra. Mit dem in Abschnitt 4.1 beschriebenen Softwaredesign lassen sich Schaltkreise zusammen setzen und diese mit beliebigen Registerzuständen initialisieren.

Phase Kickback ist ein Phänomen, welches in verschiedenen Szenarien auftritt. Die ersten beiden Szenarien verwenden dafür ein Quantenorakel zu einer unbekannten Funktion, im dritten Szenario wird ein Controlled-U Gatter verwendet. Mit der Implementierung der Anwendung kann gezeigt werden, dass auch Algorithmen die Phase Kickback verwenden simuliert werden können.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

Es gibt Konzepte die durch ihre komplexe Natur schwer zu verstehen sind. Mit einer geeigneten Darstellung lassen sich auch solche Konzepte veranschaulichen.

Quantenalgorithmen sind, trotz ihres beeindruckenden Potenzials, schwer zu verstehen. Für Schaltkreise mit vergleichbar wenigen QuBits, ist das Würfeldiagramm solch eine geeignete Visualisierung. Dies ergibt sich aus dem Erfolg der öffentlichen Kurse von Prof. Dr. B. Just, bei open HPI, *Quantum Computing*. In diesen Kursen wird das Würfeldiagramm eingesetzt, um Quantenalgorithmen anschaulich darzustellen.

Die schnell fortschreitende Entwicklung im Bereich Quantencomputing zeigt, dass es notwendig ist in der Lehre zunehmend auf diese Themen einzugehen. Das Bundesministerium für Bildung und Forschung finanziert einige Projekte im Bereich Quantencomputing und Lehre.[1]

Diese Arbeit präsentiert eine plattformunabhängige Implementierung eines Quantensimulators, der jedes Zwischenergebnis, in Form des Würfeldiagramms, sowie die Übergänge zwischen den Ergebnissen darstellt.

Neben der Implementierung die zu dieser Arbeit gehört, gibt es noch weitere Simulatoren. Verglichen werden hier fünf Simulatoren:

1. QuantenLernApp (QLA) Implementierung dieser Arbeit
2. Fraunhofer-Institut für Graphische Datenverarbeitung IDG[2]
3. QuanTUK[3]
4. Algassert Quirk[4]
5. Qiskit Bibliothek für Python[5]

Tabelle 1.1 zeigt welche Features in den genannten Simulatoren zur Verfügung stehen. Qiskit läuft als Industriestandard ohne Konkurrenz, lediglich die Würfeldarstellung ist mit Qiskit nicht möglich.

Diese Arbeit ist in fünf Abschnitte aufgeteilt. Zunächst werden in Kapitel 2 die Grundlagen des Quantencomputing beschrieben. Dabei wird erläutert worin sich

[1] BMBF, *Projekte - Quantentechnologien*.
[2] *QCVIS*.
[3] QuanTUK, *DCN_Webtool*.
[4] *Quirk: Quantum Circuit Simulator*.
[5] IBM, *Qiskit*.

Feature	QLA	QCVIS	QuanTUK	Quirk	Qiskit
Gatter H, X, Y, Z	✓	✓	✓	✓	✓
Gatter CNOT, Toffoli	✓	(✓)	(✓)	✓	✓
Rotationsgatter	×	×	✓	✓	✓
Quantenorakel aus Funktion	✓	×	×	(✓)	✓
Controlled-U Gatter	✓	(✓)	×	✓	✓
Komplexe Initialwerte	✓	×	✓	(✓)	(✓)
Schaltkreis Darstellung	×	✓	×	✓	✓
Würfeldiagramm	✓	(✓)	(✓)	×	×
bis 3 QuBits	✓	✓	✓	✓	✓
bis 4 QuBits	✓	✓	×	✓	✓
mehr als 4 QuBits	×	✓	×	✓	✓
OS unabhängig	✓	✓	✓	✓	(✓)
für mobile Geräte	✓	(✓)	(✓)	(✓)	×
Offline	✓	×	×	×	✓

✓ Vollständig möglich. (✓) Eingeschränkt möglich. × Nicht möglich.

Tabelle 1.1.: Übersicht der Features verschiedener Quantensimulatoren

QuBits von klassischen Bits unterscheiden, wie Quantengatter auf diesen angewendet werden, und wie sich das Messen eines QuBits auf ein Register auswirkt.

In Kapitel 3 werden Anforderungen definiert, die erfüllt werden müssen, damit die Anwendung als erfolgreich implementiert gilt. Diese Anforderungen gliedern sich in zwei Bereiche, zum einen werden funktionale Anforderungen beschrieben. Diese definieren welche Funktionen bereitstehen müssen, damit die Anwendung sinnvoll eingesetzt werden kann. Zum anderen werden Anforderungen definiert, welche sicherstellen, dass die Anwendung in Zukunft weiter entwickelt werden kann.

Kapitel 4 beschreibt den Aufbau der Anwendung. UML-Diagramme der wichtigsten Klassen zeigen, wie sich die Komponenten der Anwendung zusammen setzen.

In Kapitel 5 wird das Phänomen Phase Kickback beschrieben, dieses tritt in drei Szenarien auf. Die verschiedenen Szenarien werden dabei einzeln vorgestellt.

In Kapitel 6 wird überprüft, ob die Anforderungen umgesetzt sind, sowie evaluiert, inwiefern Phase Kickback simuliert und dargestellt werden kann.

2. Quantencomputing Basics

Diese Kapitel gibt einen Überblick über die Grundlagen des Quantencomputing. Dabei wird sowohl die mathematische Seite betrachtet, als auch die Darstellung präsentiert, die einen visuellen Zugang bereitet. Zu beachten ist, dass hier nicht auf die physikalischen Grundlagen eingegangen, sondern lediglich die mathematische Modellvorstellung beschrieben wird.

2.1. QuBits im Gegensatz zu klassischen Bits

Ein Bit ist die kleinste Dateneinheit für einen Computer. Solche Bits unterliegen zwei Eigenschaften, die so grundlegend und selbstverständlich sind, dass diese oftmals nicht aufgeführt werden. Die erste Eigenschaft ist der Realismus, diese besagt, dass ein gegebenes Bit zu jedem Zeitpunkt einen bestimmten messbaren Wert hat, auch wenn der Wert noch nicht bekannt ist. Zusätzlich bleibt der Wert durch die Messung unverändert. Die zweite Eigenschaft Lokalität beschreibt, dass Änderungen an einem Bit keine direkten Auswirkungen auf andere Bits haben. Die Auswirkungen werden über ein physikalisches Medium, mit maximal der Geschwindigkeit des Lichts übertragen.[1]

Für einen Quantencomputer ist das entsprechende Äquivalent ein Quantenbit oder QuBit. Wobei sowohl die Eigenschaft des Realismus, als auch die Lokalität nicht gelten. Befindet sich ein QuBit in einem überlagerten Zustand, ist bis zur Messung unbestimmt, in welchem Zustand sich ein QuBit befindet. Zusätzlich ändert sich der Zustand des QuBit durch die Messung. Auf welche Weise die Messung den Zustand eines QuBit verändert wird in Kapitel 2.4 erläutert.

Für ein QuBit entfällt in bestimmten Situationen auch die zweite Eigenschaft. Die Messung eines QuBit eines sogenannten „verschränkten“ Systems beeinflusst den Zustand der anderen QuBits dieses Systems. Dabei wirkt sich die Zustandsänderung durch die Messung unmittelbar, also mit mehr als Lichtgeschwindigkeit aus.[2] Eine ausführliche Beschreibung verschränkter Systeme ist in Kapitel 2.2 zu finden.

[1] Siehe Just, *Quantencomputing kompakt*, S. 1f.
[2] Siehe Yin et al., „Bounding the Speed of ‘spooky Action at a Distance’“.

QuBits haben im Gegensatz zu klassischen Bits nicht den Wertebereich $\{0, 1\}$, sondern werden in der Ket-Notation angegeben. Ein QuBit hat die Gestalt

$$|x\rangle = x_0 \cdot |0\rangle + x_1 \cdot |1\rangle = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \tag{2.1}$$

$|0\rangle$ und $|1\rangle$ sind dabei die Basiszustände. x_0 und x_1 werden als Amplituden bezeichnet, wobei x_0 und x_1 Komplexe Zahlen sind, deren Betragsquadrate summiert 1 ergeben. Grund dafür ist, dass die Betragsquadrate jeweils die Wahrscheinlichkeiten für das Eintreten des Messergebnisses $|0\rangle$ beziehungsweise $|1\rangle$ angeben. Die Summe aller Wahrscheinlichkeiten muss dann 1 ergeben.

Wenn sowohl x_0, als auch x_1, nicht 0 sind, spricht man von einem überlagerten Zustand oder von Superposition. Ein bekanntes Beispiel für einen überlagerten Zustand ist das Gedankenexperiment „Schrödingers Katze“.[3]

2.2. Quantenregister

Werden QuBits nicht mehr einzeln betrachtet, sondern als Teile eines Systems, wird dieses als Register bezeichnet. Um den Zustand zweier unabhängiger QuBits als Register zu beschreiben, werden die Zustandsvektoren mit dem Kronecker Produkt ausmultipliziert.

$$\begin{aligned} |x\rangle &= x_0 |0\rangle + x_1 |1\rangle \\ |y\rangle &= y_0 |0\rangle + y_1 |1\rangle \end{aligned}$$

$$\begin{aligned} |x\rangle \otimes |y\rangle &= (x_0 |0\rangle + x_1 |1\rangle) \otimes (y_0 |0\rangle + y_1 |1\rangle) \\ &= x_0 y_0 |00\rangle + x_0 y_1 |01\rangle + x_1 y_0 |10\rangle + x_1 y_1 |11\rangle \\ &= |xy\rangle \end{aligned}$$

Für ein Register mit drei QuBits ergibt sich als Darstellung:

$$|xyz\rangle = \alpha_0 |000\rangle + \alpha_1 |001\rangle + \alpha_2 |010\rangle + \alpha_3 |011\rangle + \cdots + \alpha_7 |111\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \cdots \\ \alpha_7 \end{pmatrix} \tag{2.2}$$

Bei einem Register mit n QuBits ergeben sich 2^n Amplituden α_0 bis α_{2^n-1}, für die zu jedem Zeitpunkt gilt: Die Summe der Betragsquadrate der Amplituden ist gleich 1.

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1 \tag{2.3}$$

[3] Siehe. Schrödinger, „Die gegenwärtige Situation in der Quantenmechanik“, S. 812.

Eine grafische Darstellung für Registerzustände wird in Abschnitt 2.3 beschrieben. Lässt sich ein Registerzustand als Tensorprodukt der Zustände einzelner QuBits schreiben, so heißen die QuBits „unverschränkt“ und der Registerzustand „separabel“. Es gibt aber auch Beispiele verschränkter QuBits, also nicht separabler Zustände. Im Falle von zwei QuBits bedeutet das, dass die Gleichung 2.4 keine Lösung x_0, x_1, y_0, y_1 besitzt.

$$(x_0 |0\rangle + x_1 |1\rangle) \otimes (y_0 |0\rangle + y_1 |1\rangle) = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle \tag{2.4}$$

Lassen sich die QuBits nicht mehr einzeln betrachten, befindet sich das Register $|xy\rangle$ in diesem Fall in einem verschränkten Zustand.[4] Ein Beispiel für einen Registerzustand, der sich nicht zerlegen lässt, ist $\frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle$. Dieser Zustand ist bekannt unter dem Namen EPR-Paar. Benannt nach den Physikern Einstein, Podolsky und Rosen, welche diesen 1935 als paradox beschreiben.[5]

2.3. Würfeldiagramm

Für die Darstellung von QuBits gibt es verschiedene Möglichkeiten. Eine weit verbreitete Darstellung ist die Bloch-Kugel, bei der der Zustand eines QuBit als Vektor in einer Kugel mit Radius 1 eingezeichnet ist. Um ein Register mit zwei oder mehr verschränkten QuBits darzustellen, wird für jedes QuBit eine separate Bloch-Kugel gezeichnet, wie in Abbildung 2.1 zu sehen ist. Bei dieser Darstellung ist nicht direkt erkennbar, ob es sich um einen verschränkten Zustand handelt.

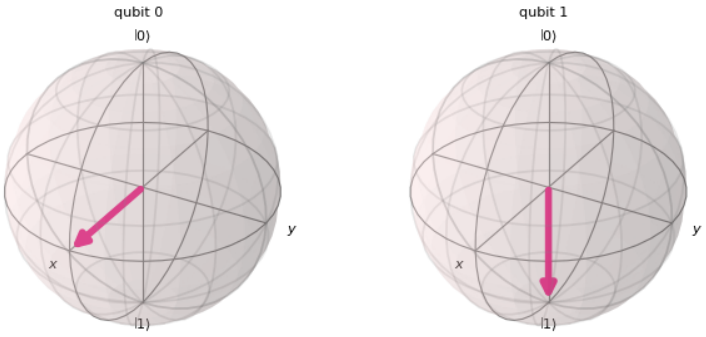


Abbildung 2.1.: Bloch-Kugel Darstellung für Register mit zwei QuBits[6]

[4] Just, *Quantencomputing kompakt*.
[5] Siehe Einstein, Podolsky und Rosen, „Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?“

Eine Darstellung, bei der auch verschränkte Zustände als solche erkennbar sind, wird in Just, *Quantencomputing kompakt* erstmals angewendet und wird inzwischen als „Würfeldiagramm“ oder „cube model for quantum computing“ bezeichnet.

Ein einzelnes QuBit wird als Linie dargestellt, an deren beiden Enden jeweils ein Quadrat der Kantenlänge 1 liegt.

In den Quadraten werden die Amplituden eingezeichnet, die Positionierung der Quadrate zeigt an, zu welchem Basiszustand die Amplitude gehört.

Von der unteren linken Ecke jedes Quadrats geht ein Pfeil gegen den Uhrzeigersinn entlang der Kante.

Die Darstellung der Amplituden folgt der Darstellung komplexer Zahlen in Feynman, *Quantenelektrodynamik*. Dabei entspricht die Länge der Pfeile den Beträgen der Amplituden. Der Flächeninhalt des inneren Quadrats entspricht dem Amplitudenquadrat, also der Wahrscheinlichkeit. Durch eine Rotation der Quadrate kann die Phase dargestellt werden. In Abbildung 2.2 ist ein QuBit dargestellt, welches sich im Zustand aus Gleichung 2.5 befindet.

$$|\psi\rangle = \sqrt{\frac{1}{4}} \cdot e^{i \cdot 90°} \cdot |0\rangle + \sqrt{\frac{3}{4}} \cdot e^{-i \cdot 135°} \cdot |1\rangle \tag{2.5}$$

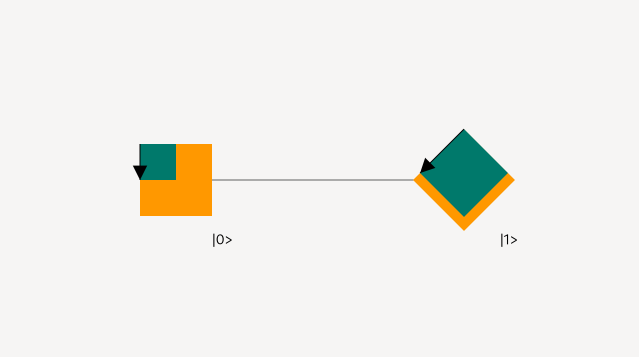


Abbildung 2.2.: Ein QuBit Register mit komplexen Amplituden als Würfeldiagramm

Bei einem Register mit zwei QuBits wird das zweidimensionale Analogon einer Linie verwendet, ein Quadrat, in dessen Ecken die bereits bekannten kleinen Quadrate sitzen. Das erste QuBit breitet sich entlang der Rechts-Links-Achse aus, das zweite entlang der Oben-Unten-Achse, wie in Abbildung 2.3 zu sehen ist.

So lässt sich dieses Diagramm auch auf Register mit drei QuBits als Würfel ergänzen, mit der Vorne-Hinten-Achse für das dritte QuBit, Abbildung 2.4 links. Und einem Tesserakt mit der Ana-Kata-Achse[7] für das vierte QuBit, Abbildung 2.4 rechts.

[6] *Qiskit.Visualization.Plot_bloch_multivector — Qiskit 0.43.2 Documentation*
[7] Siehe Hinton, *The Fourth Dimension*, S. 160.

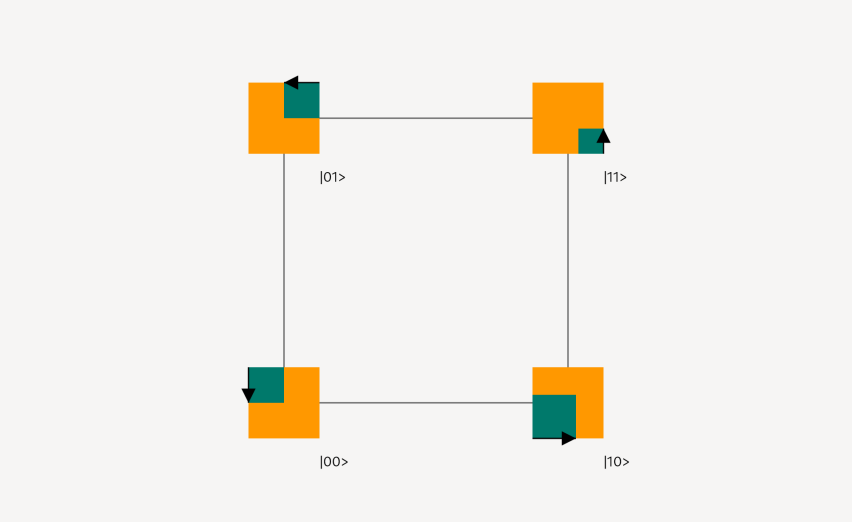


Abbildung 2.3.: Zwei-QuBit-Register mit komplexen Amplituden als Würfeldiagramm

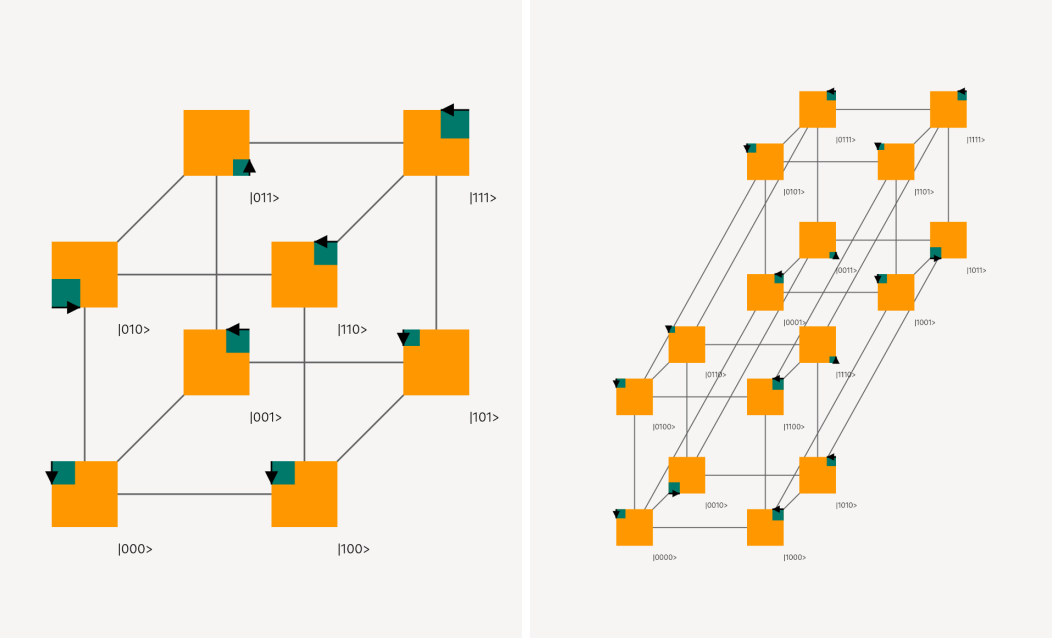


Abbildung 2.4.: Drei- und vier-QuBit-Register mit komplexen Amplituden als Würfeldiagramm

Auch sind weitere Dimensionen denkbar, eine Darstellung dieser wird jedoch etwas unübersichtlich. Für n QuBits ist die geometrische Form, jeweils das n-dimensionale Objekt, welches 2^n Ecken hat, in denen n Kanten im rechten Winkel zusammenlaufen.

2.4. Messung eines QuBits

Wie schon in Abschnitt 2.1 beschrieben, hat das Messen eines QuBit eine Änderung des Zustands zur Folge. Betrachtet man ein QuBit im Zustand

$$|\psi\rangle = \alpha_0 \cdot |0\rangle + \alpha_1 \cdot |1\rangle \tag{2.6}$$

wird eine Messung, mit Wahrscheinlichkeit α_0^2 das Ergebnis 0 ergeben. Das QuBit befindet sich danach im Zustand $|0\rangle$. Äquivalent dazu wird mit Wahrscheinlichkeit α_1^2 das Ergebnis $|1\rangle$ gemessen und das QuBit befindet sich im Zustand $|1\rangle$.

Anders als bei einem Zufallsexperiment in der klassischen Mechanik, bei der eine Münze schon auf der einen oder anderen Seite liegt, auch wenn noch nicht nachgesehen wurde, hat ein QuBit bis zum Zeitpunkt des Messens noch keinen der beiden möglichen Werte angenommen.[8]

In Abbildung 2.5 ist die Auswirkung eines Messvorgangs des ersten QuBit auf einem vier QuBit Register dargestellt. Vor dem Messen hat das Register den Zustand, der in Gleichung 2.7 beschrieben ist.

$$\begin{aligned} |yx_1x_2x_3\rangle = &\sqrt{\frac{1}{8}} \cdot |0001\rangle + \sqrt{\frac{1}{8}} \cdot |0010\rangle + \sqrt{\frac{1}{8}} \cdot |0011\rangle \\ &+ \sqrt{\frac{1}{8}} \cdot |0100\rangle + \sqrt{\frac{1}{8}} \cdot |0101\rangle + \sqrt{\frac{1}{8}} \cdot |0110\rangle \\ &+ \sqrt{\frac{1}{8}} \cdot |1000\rangle + \sqrt{\frac{1}{8}} \cdot |1111\rangle \end{aligned} \tag{2.7}$$

Um die Wahrscheinlichkeit zu erhalten, dass y als $|0\rangle$ gemessen wird, werden alle Amplitudenquadrate addiert, bei denen y gleich $|0\rangle$ ist. Entsprechend ist die Wahrscheinlichkeit, dass y als $|1\rangle$ gemessen wird, die Gegenwahrscheinlichkeit, also $1 - P(y = |0\rangle)$.

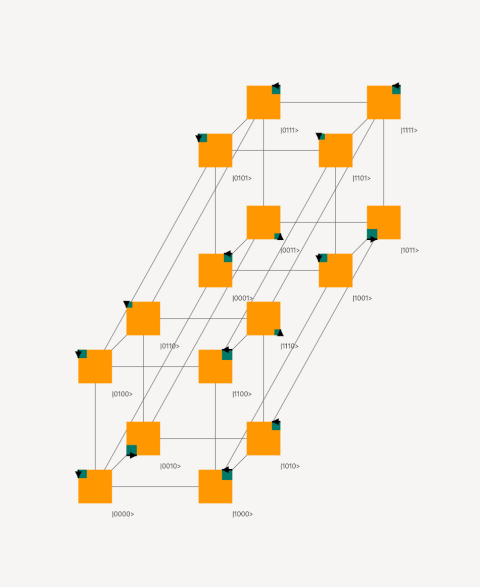
$$\begin{aligned} P(y = |0\rangle) &= \sum_{i=0000_2}^{1111_2} \begin{cases} \alpha_i^2 & \text{für } i \wedge 1000_2 = 0000_2 \\ 0 & \text{sonst} \end{cases} \\ &= \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = \frac{6}{8} \end{aligned} \tag{2.8}$$

Damit die Flächen der Quadrate wieder 1 ergeben und der Zustandsvektor normiert ist, müssen die Amplituden mit der Wurzel des Kehrwerts der Wahrscheinlichkeit multipliziert werden. Nach einer Messung mit dem Ergebnis $|0\rangle$ befindet sich das Register im Zustand

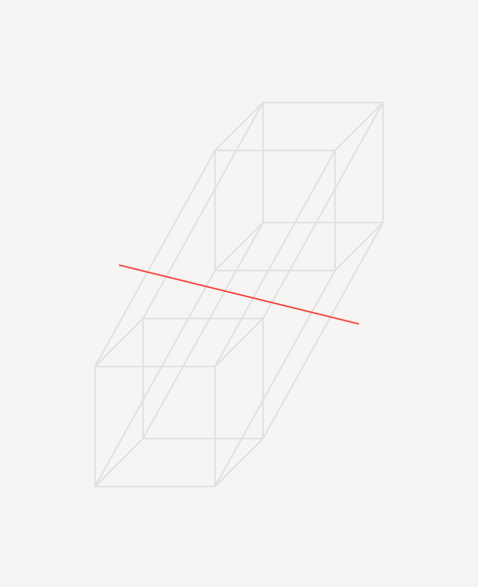
$$\begin{aligned} |0x_1x_2x_3\rangle = &\sqrt{\frac{8}{6}} \cdot (\sqrt{\frac{1}{8}} \cdot |0001\rangle + \sqrt{\frac{1}{8}} \cdot |0010\rangle + \sqrt{\frac{1}{8}} \cdot |0011\rangle \\ &+ \sqrt{\frac{1}{8}} \cdot |0100\rangle + \sqrt{\frac{1}{8}} \cdot |0101\rangle + \sqrt{\frac{1}{8}} \cdot |0110\rangle) \end{aligned} \tag{2.9}$$

[8] Senno, Strohm und Acín, *Quantifying the Intrinsic Randomness of Quantum Measurements*.

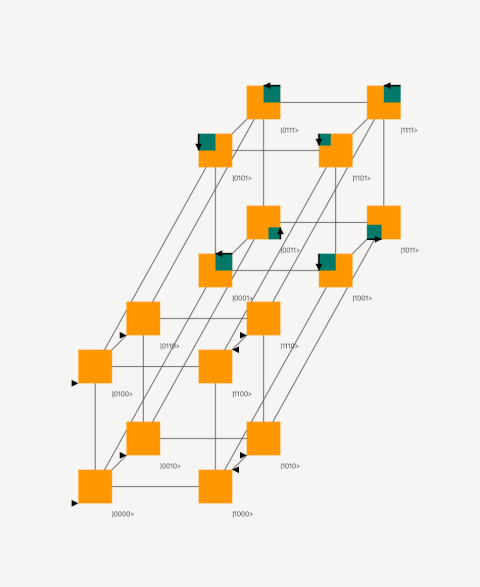
Zu sehen ist dies in Abbildung 2.5d. Äquivalent dazu ergibt sich auch der Zustandsraum für eine Messung von y als $|1\rangle$, Abbildung 2.5c.



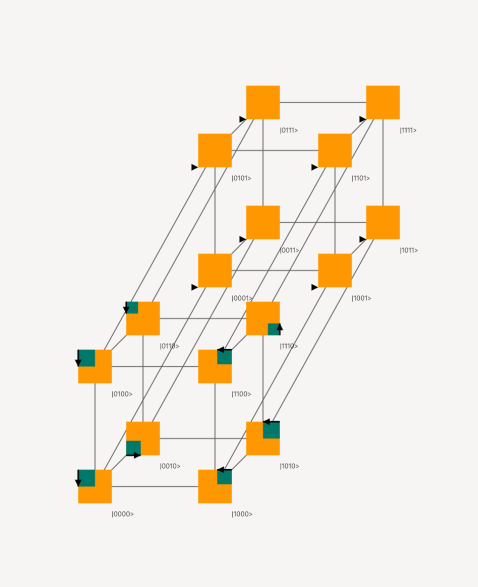
(a) Initial Zustand



(b) Auswirkung der Messung



(c) Zustand nach Ergebnis 1



(d) Zustand nach Ergebnis 0

Abbildung 2.5.: Messung des vierten QuBits eines Register mit vier QuBits

Sollen mehrere QuBits gemessen werden, kann dies als die Messung eines QuBit nach dem anderen betrachtet werden. Die Reihenfolge hat dabei keine Auswirkung auf das Endergebnis.[9]

[9]Siehe Just, *Quantencomputing kompakt*, 70f.

2.5. Schaltkreis und Quantengatter

Die grundlegenden Bausteine eines klassischen Schaltkreises sind Gatter. Eines dieser Gatter ist das NOT-Gatter, welches den Input entsprechend Tabelle 2.1 verarbeitet.

Input	Output
0	1
1	0

Tabelle 2.1.: Klassisches NOT-Gatter.

Für Quantencomputer gibt es eine Reihe eigener Quantengatter, welche die Grundlage für Quantenalgorithmen darstellen. Die wichtigsten Quantengatter sind das Pauli-X Gatter, Pauli-Y Gatter, Pauli-Z Gatter und das Hadamard Gatter.

Das Pauli-X Gatter entspricht dem NOT-Gatter des klassischen Schaltkreises. Für ein QuBit in einem Basiszustand, verhält sich das Quantengatter entsprechend Tabelle 2.2

Input	Output
$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$

Tabelle 2.2.: Pauli-X auf Basiszustände.

Allgemein werden bei einem QuBit die Amplituden vertauscht. Aus $\alpha\,|0\rangle + \beta\,|1\rangle$ wird nach Anwendung eines Pauli-X Gatters $\beta\,|0\rangle + \alpha\,|1\rangle$. Für jedes Quantengatter existiert eine unitäre Matrix. Die Matrix die das Pauli-X Gatter darstellt, ist in Gleichung 2.10 zu sehen.

$$\begin{pmatrix}\alpha\\ \beta\end{pmatrix} \cdot \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix} = \begin{pmatrix}\beta\\ \alpha\end{pmatrix} \tag{2.10}$$

Dargestellt wird Pauli-X entweder durch einen Kasten mit Aufschrift „X“ wie in Abbildung 2.6, oder in der üblicheren Darstellung wie in Abbildung 2.7.

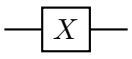


Abbildung 2.6.: Pauli-X als Quantengatter

Das Pauli-Z Gatter und das Hadamard Gatter werden analog zu Abbildung 2.6 mit den Buchstaben „Z“ und „H“ dargestellt.

Abbildung 2.7.: Pauli-X in üblicher Darstellung

Die Auswirkung von Pauli-X auf einen beliebigen Zustand ist in Abbildung 2.8 zu sehen. Die unitären Matrizen dieser und weiterer Quantengatter finden sich im Anhang in Abschnitt I.

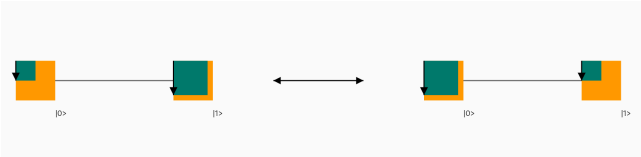


Abbildung 2.8.: Würfeldiagramm mit Übergang für Pauli-X Gatter

Als Schaltkreis wird ein Register mit beliebig vielen QuBits bezeichnet, auf welchen beliebige Quantengatter angewendet werden. In Abbildung 2.9 ist ein beispielhafter Schaltkreis zu sehen, der keine besondere Funktion erfüllt. Ziel ist es, einen Schaltkreis so zu entwerfen, dass dabei eine sinnvolle Aufgabe gelöst wird.

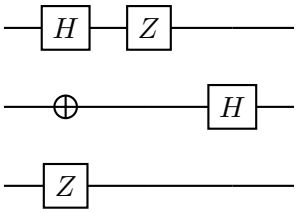


Abbildung 2.9.: Beispiel Schaltkreis mit drei QuBits

Neben den bereits bekannten Quantengatters, welche jeweils ein Input QuBit und ein Output QuBit haben, gibt es auch Quantengatter, welche auf mehreren QuBits arbeiten.

Ein Beispiel hierfür ist das CNOT-Gatter, auch Controlled NOT genannt. Dargestellt wird CNOT wie in Abbildung 2.10.

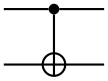


Abbildung 2.10.: Darstellung CNOT-Gatter

2.6. Matrizendarstellung von Schaltkreisen

Wie schon in Abschnitt 2.5 beschrieben, lassen sich die Quantengatter als Matrizen interpretieren. Ein wichtiger Bestandteil, der noch fehlt, ist das Identitätsgatter. Die entsprechende Matrix zu diesem trivialen Quantengatter ist die Einheitsmatrix der Größe 2.

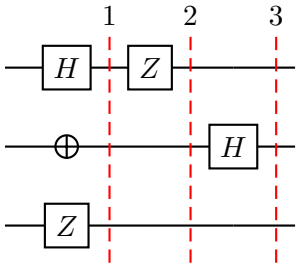


Abbildung 2.11.: Beispiel Schaltkreis mit drei QuBits eingeteilt in drei Schritte

Um den Zustandsvektor nach einer Operation zu erhalten, wird zunächst die Matrix U_n berechnet. Diese ergibt sich aus dem Tensorprodukt der angewendeten Quantengatter. Zu beachten ist, dass QuBits auf denen kein spezifisches Quantengatter angewendet wird, mit dem Identitätsgatter aufgefüllt werden.

$$U_1 = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.11}$$

$$U_2 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.12}$$

$$U_3 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.13}$$

In den Gleichungen 2.11 bis 2.13 werden die Matrizen beschrieben, die den Schritten aus Abbildung 2.11 entsprechen.

$$(U_3 \cdot (U_2 \cdot (U_1 \cdot V_{initial}))) = (U_3 \cdot U_2 \cdot U_1) \cdot V_{initial} = V_{final} \tag{2.14}$$

Wie in Gleichung 2.14 zu erkennen ist, ergibt sich in beiden Fällen der gleiche finale Zustandsvektor. Ob jeder Schritt separat mit dem vorhergehenden Zustandsvektor multipliziert wird, oder ob erst die unitären Matrizen aller Schritte multipliziert werden, ist im Ergebnis gleich, weil die Matrixmultiplikation assoziativ ist. So lässt sich ein Schaltkreis als eine Matrix beschreiben.

3. Anforderungen

In diesem Kapitel werden die grundlegenden Anforderungen definiert, welche die Anwendung erfüllen soll. In Abbildung 3.1 ist zu erkennen, wie die Anwendung in der Lehre eingesetzt werden kann. Das Diagramm beschreibt die Systemlandschaft entsprechend dem C4 Model.[1] Eine ausführliche Beschreibung der Funktionalität, sowie ein Mockup der Benutzeroberfläche findet sich in Abschnitt 3.1.

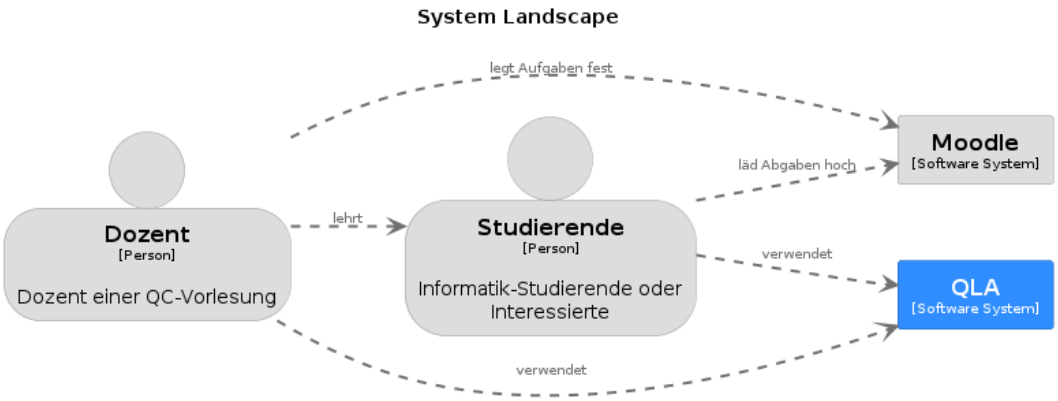


Abbildung 3.1.: System Landscape Diagramm der Quanten Lern App nach C4 Model

Die funktionalen Anforderungen werden in Abschnitt 3.2 beschrieben. Zusätzlich werden Prioritätsstufen für die funktionalen Anforderungen festgelegt. Anforderungen, welche die Qualität beschreiben, werden in Abschnitt 3.3 angeführt.

3.1. Grundlegende Funktionsbeschreibung

Entwickelt wird eine Anwendung, die plattformunabhängig einen Quantenschaltkreis Schritt für Schritt darstellt. Die Anwendung soll in der Lehre eingesetzt werden und hauptsächlich von Studierenden der Informatik, Physik oder angrenzender Disziplinen verwendet werden. Auch die Dozenten sollen die Anwendung in der Vorbereitung nutzen können.

Schaltkreise, die als Datenstruktur vorliegen werden schrittweise berechnet. Jeder Zwischenschritt soll dargestellt werden, um ein Verständnis des Schaltkreises für

[1] *The C4 Model for Visualising Software Architecture*.

die Studierenden zu ermöglichen. Die Darstellung soll in Form des Würfeldiagramms erfolgen, welches in Abschnitt 2.3 beschrieben ist. Darüber hinaus sollen auch andere Darstellungsformen ermöglicht werden.

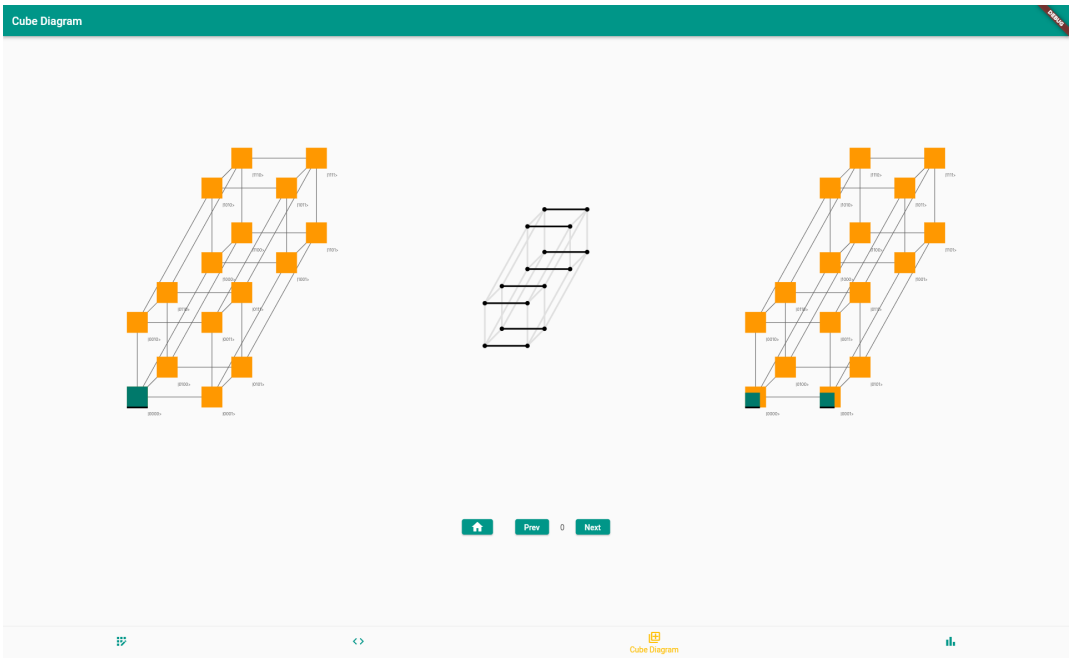


Abbildung 3.2.: Screenshot der Visualisierung eines vier QuBit Registers und einer Hadamard Transformation auf dem ersten QuBit.

Für diese Arbeit ist neben dem Simulator und dem Softwaredesign ausschließlich eine Ansicht relevant. Wie in Abbildung 3.2 zu sehen ist, soll jeweils links der Zustand vor einer Operation und rechts der Zustand nach dieser abgebildet werden. In der Mitte soll dargestellt werden, welche Auswirkungen die Operation hat.

Um die Plattformunabhängigkeit zu erreichen, wird die Anwendung mit dem Flutter Framework entwickelt. Die Vorteile des Frameworks sind, dass aus einer Codebase native Anwendungen für mehrere Plattformen erzeugt werden können.[2]

Mit der Darstellung des Würfeldiagramms ist die Anwendung bisher einzigartig. Vergleichbare Anwendungen scheitern an der korrekten Umsetzung des Diagramms, oder können nur das Endergebnis darstellen. Lediglich der Simulator des Fraunhofer-Instituts IDG QCVIS[3] stellt auch Zwischenergebnisse in einer dem Würfeldiagramm ähnlichen Form dar. Dieser Simulator lässt sich jedoch nicht ohne Internetverbindung verwenden.

Im Rahmen dieser Arbeit soll die Simulation und Darstellung von Schaltkreisen bearbeitet werden. Die Konfiguration von Schaltkreisen innerhalb der Anwendung

[2]Flutter Team, *Flutter - Build Apps for Any Screen*.
[3]QCVIS

wird aus Zeitgründen nicht entwickelt. Ziel ist die Anwendung so zu entwickeln, dass diese und andere Feature in Zukunft ergänzt werden können. Einen Überblick über denkbare zukünftige Features findet sich im Anhang Abschnitt II.

3.2. Funktionale Anforderungen

Als Leitfaden für den Entwicklungsprozess sollen im Folgenden Anforderungen definiert werden, die für die Funktion der Anwendung relevant sind. Die Anforderungen werden in drei Prioritätsstufen eingeteilt:

1. **Must-have**: Funktionen, die notwendig für die Funktionsweise der Anwendung sind.

2. **Usefull**: Funktionen, die einen Mehrwert über die Grundfunktionen bieten.

3. **Nice to have**: Funktionen, die voraussichtlich über das zeitliche Limit hinaus gehen.

Must-have Funktionen

Zu den Anforderungen mit der höchsten Priorität zählt, dass sich ein Quantenschaltkreis aus den dafür notwendigen Komponenten zusammen setzen lässt. Dabei soll die Möglichkeit bestehen, dass unterschiedlich viele QuBits verwendet und diese mit komplexen Zahlen initialisiert werden können.

Zusätzlich sollen auch beliebige Kombinationen an Quantengatters dem Schaltkreis hinzugefügt werden können. Anzumerken ist, dass Schaltkreise zunächst nur als Datenstruktur innerhalb des Programmcodes instanziiert werden sollen. Der Aufbau eines Schaltkreises aus der Benutzeroberfläche ist nicht Teil dieser Arbeit.

Eine weitere wichtige Anforderung ist der Simulator, welcher einen Schaltkreis übergeben bekommt und die berechneten Zwischenschritte als Stream zurückgibt. Dabei ist darauf zu achten, dass die Berechnungen nicht dazu führen, dass die Benutzeroberfläche nicht mehr auf Eingaben reagieren kann. Der Simulator muss also unabhängig des Event Loop ausgeführt werden.[4]

Die letzte zentrale Anforderung liegt darin, die berechneten Ergebnisse des Simulators darzustellen. Dafür soll ein weiteres Plugin angelegt werden, welches die notwendigen Klassen zur Visualisierung des Statevektors, sowie der Übergänge bereitstellt.

[4] Bhat, *Multithreading in Flutter Using Dart Isolates*.

Usefull Funktionen

Eine nützliche Funktion kann es sein, Konstruktoren für gängige Quantengatter zu haben. Wenn in einem Schaltkreis ein Pauli-X Gatter verwendet wird, muss die Matrix nicht neu angegeben werden, lediglich der Index des QuBit, auf dem das Quantengatter angewendet werden soll. Zu den Quantengattern mit häufiger Verwendung zählen neben Pauli-X, auch Pauli-Y, -Z und das Hadamard Gatter.

Ein Parser für boolesche Funktionen in Form einer LaTeX-ähnlichen Domänen spezifischen Sprache, zu implementieren, um daraus die Quantenorakel aufzubauen, ist nicht zwingend für die Verwendbarkeit der Anwendung notwendig, kann aber unter Umständen nützlich sein.

Schaltkreise nicht nur im Programmcode aufzubauen, sondern als Datei zu speichern, und wieder einzulesen, kann die Verwendbarkeit der Anwendung verbessern. So kann die Anwendung schon dann in der Lehre eingesetzt werden, wenn das Aufbauen eines Quantenschaltkreises noch nicht implementiert ist.

Nice to have Funktionen

Eine Funktion, die den Rahmen dieser Arbeit vermutlich überschreitet, ist es Quantenschaltkreise innerhalb der Benutzeroberfläche zusammen setzen zu können, als Vorlage dient hier der Quantensimulator des Fraunhofer-Institutes für Graphische Datenverarbeitung IDG.[5]

Die Quantengatter RX, RY und RZ sowie das T-Gatter sowohl simulieren als auch darstellen zu können ist eine weitere Funktion, die den Rahmen dieser Arbeit überschreiten wird, für die zukünftige Entwicklung jedoch in Betracht gezogen werden kann.

Eine Funktion, die die Usability der Anwendung verbessern kann, ist ein In-App-Tutorial, bei dem alle wichtigen Funktionen der Anwendung vorgestellt werden. Eine Implementierung dieser Funktion ist jedoch für einen Proof of Concept nicht notwendig und wird aus diesem Grund auf einen späteren Zeitpunkt verschoben.

3.3. Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen gliedern sich in drei Bereiche. Zum einen ist eine ausführliche Dokumentation aller öffentlicher Schnittstellen notwendig, um sicherzustellen, dass zukünftig an der Anwendung weiter entwickelt werden kann. Zur Dokumentation wird Dartdoc[6] eingesetzt. Dartdoc kann sowohl für Flutter spezifische, als auch Flutter unabhängige Bereiche der Anwendung verwendet werden.

[5] *QCVIS*.
[6] *Dart Doc*.

Zu beachten ist, dass nur die Teile der Anwendung dieser Anforderung unterliegen, welche im Rahmen dieser Arbeit entstehen.

Eine weiter qualitative Anforderung besteht in der Testbarkeit der Anwendung. Mit dem Flutter Framework wird eine Testumgebung mitgeliefert, diese unterscheidet Tests in drei Stufen:[7]

1. **Unit-Tests**, für einzelne Funktionen und Methoden, wie Matrixmultiplikation.
2. **Widget-Tests**, die Komponenten der UI werden korrekt angelegt.
3. **Integration-Tests**, Systemtests die auf einem automatisierten Androidsimulator ausgeführt werden.

Integration-Tests werden in Flutter nur benötigt, wenn im Rahmen von End-to-End-Tests eine Verbindung mit einem Server notwendig ist. Aus diesem Grund wird die Anwendung ausschließlich durch die ersten beiden Teststrategien überprüft.

Eine zusätzliche Teststrategie ist der Golden-Test, welcher eine Form des Widget-Tests ist. Bei diesen Tests wird das Widget gerendert und mit einem Bild des zu erwartenden Ergebnisses abgeglichen. Für die Diagramme ist diese Form der Tests gut geeignet.

Um die Testbarkeit und die zukünftige Entwicklung an der Anwendung zu erleichtern, soll der Simulator als eigenständiges Plugin entwickelt werden.

Performanz ist ein wichtiges Kriterium, wenn es um mobile Anwendungen geht. Als harte Anforderung wird festgelegt, dass die Benutzeroberfläche auch zum Zeitpunkt der Berechnung auf Eingaben reagieren muss.

[7] Flutter Team, *Testing Flutter Apps*.

4. Softwaredesign

In diesem Kapitel wird der Aufbau der Anwendung beschrieben, dabei wird großteils auf den Simulator und die damit verbundenen Bereiche eingegangen. Auf den grundlegenden Aufbau einer Flutter Anwendung wird dabei nicht zusätzlich eingegangen. Ein grober Überblick darüber, wie die einzelnen Komponenten zusammenpassen, ist in Abbildung 4.1 zu sehen. Wobei der Quantensimulator in Abschnitt 4.1 und 4.2 beschrieben ist, sind in Abschnitt 4.3 und 4.4 die wichtigsten Widgets der UI beschrieben. Der Austausch der Daten wird dabei von der Statemanagement Komponente realisiert, diese ist entsprechend des BLoC Pattern umgesetzt.[1] Abschnitt 4.5 geht dabei genauer auf die Vorteile des BLoC Patterns ein.

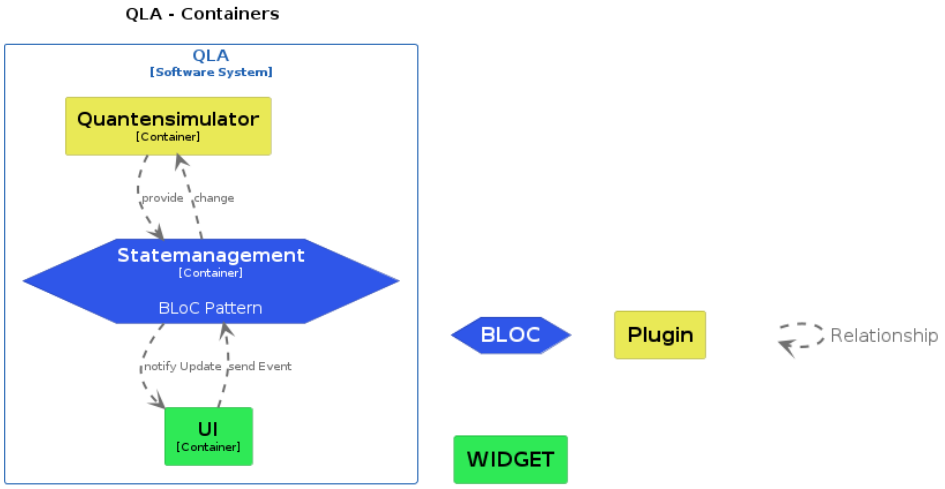


Abbildung 4.1.: Container Diagramm nach C4 Model

4.1. Schaltkreis als Datenstruktur

In diesem Kapitel wird beschrieben, wie eine Datenstruktur aufgebaut wird, die einen Quantenschaltkreis abbildet.

Hierfür sind maßgeblich drei Komponenten relevant, wie in Abbildung 4.2 zu sehen ist. Zum einen die Anzahl der QuBits, die im Schaltkreis verwendet werden. Die

[1] Siehe *Bloc State Management Library*.

Anzahl der QuBits ist für die Laufzeit eines Schaltkreises konstant, da wie in Kapitel 2 beschrieben, weder durch eine Messung, noch durch die Anwendung eines Quantengatters die Anzahl der QuBits verändert wird.

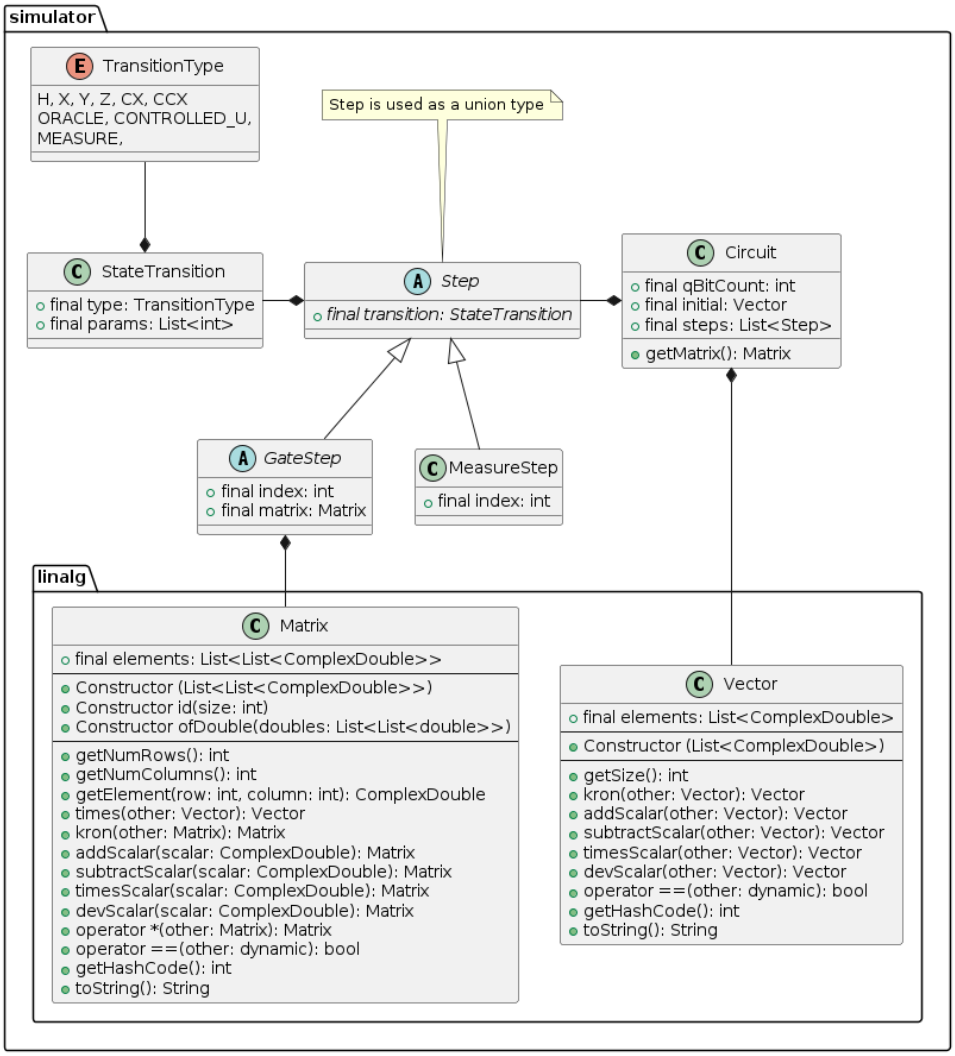


Abbildung 4.2.: Ausschnitt aus UML Diagramm für Schaltkreise

Die zweite Komponente ist der Initialwert des Registers. Hierfür wird ein *Vektor* aus dem *linalg* Paket verwendet. Somit lässt sich die Anzahl der QuBits anhand der Größe des Vektors berechnen.

Die komplexeste Komponente ist die Liste von Instruktionen. Dadurch, dass sich das Messen eines QuBits maßgeblich von der Anwendung eines Quantengatters unterscheidet, wird hier mithilfe einer abstrakten Klasse *Step* ein Union-Type geschaffen. Für eine Messung ist nur der Index des zu messenden QuBit notwendig. Bei einem Gatter kommt noch eine Matrix hinzu die, wie in Abschnitt 2.5 beschrieben, das Quantengatter abbildet.

4.2. Statevektor Simulator

In diesem Kapitel wird der Aufbau des *Quantensimulator* Plugins beschrieben, darüber hinaus wird die Funktionsweise des *Quantensimulators* dargestellt. Ein UML Diagramm der zentralen Klassen dieses Plugins ist in Abbildung 4.3 zu sehen.

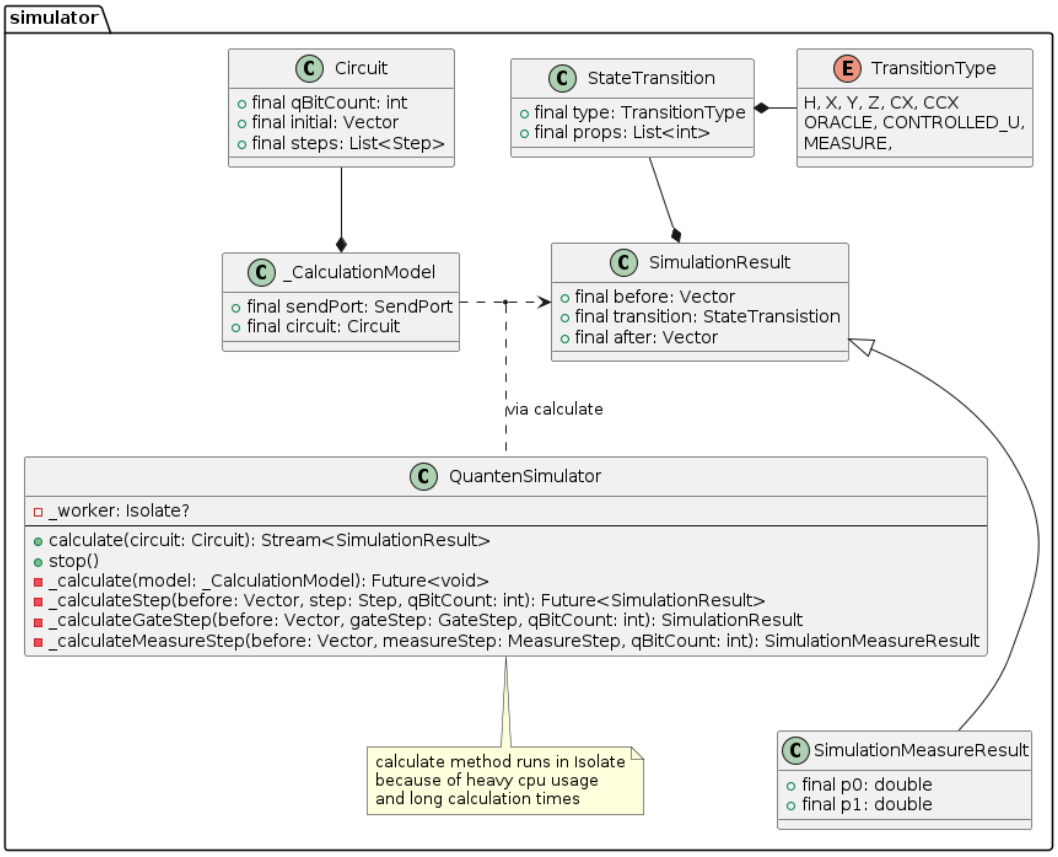


Abbildung 4.3.: Ausschnitt aus UML Diagramm für den Quantensimulator

Als Ausgangspunkt wird die Schaltkreisdatenstruktur angenommen, die in Form der *Circuit* Klasse ein Teil des Plugins dargestellt. Diese wird von der *QuantenSimulator* Klasse in ein *_CalculationModel* verpackt. Dieser Schritt ist notwendig, um zusätzlich zum Schaltkreis noch eine *SendPort* Instanz übergeben zu können. *SendPort* wird von der Dart SDK mitgeliefert und ermöglicht den Austausch von Daten zwischen verschiedenen Isolates.

Das *Worker-isolate* geht einen Schritt des Schaltkreises nach dem anderen durch und berechnet, wie in Kapitel 2 beschrieben, die Zwischenergebnisse. Kommt das *Worker-isolate* zu einem Zwischenergebnis, wird dieses über den *SendPort* dem Main Thread über die Event-Loop mitgeteilt und bereitgestellt.[2] Von dort werden die Ergebnisse

[2]Bhat, *Multithreading in Flutter Using Dart Isolates*.

als Stream an den Aufrufer der *calculate* Methode des *QuantenSimulator* weiter gereicht.

Für die Berechnung der Zwischenschritte unterscheidet der Simulator zwei Fälle. Zum einen die Messinstruktionen, die in der Methode *_calculateMeasureStep* entsprechend Abschnitt 2.4 umgesetzt werden. Zum anderen alle Quantengatter, welche jeweils eine spezifische unitäre Matrix bereitstellen. Ein Berechnungsschritt, der ein Quantengatter zur Grundlage hat, wird in der privaten Methode *_calculateGateStep* ausgeführt.

4.3. Darstellung des Statevektor

Die Darstellung eines Statevektors wird durch die von Flutter bereitgestellte API des *CustomPainter* ermöglicht. Wie in Abbildung 4.4 zu erkennen ist, erbt die *StateVectorCube* Klasse von Widget.

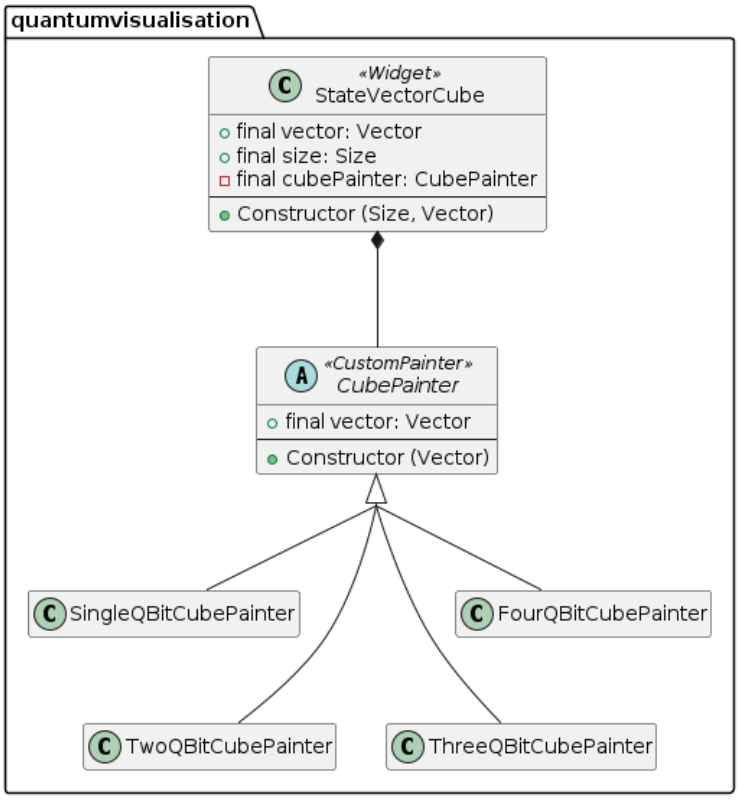


Abbildung 4.4.: UML Diagramm der Statevektor Visualisierung

Im Gegensatz zu Widgets, die durch das Framework bereitgestellt werden, hat *StateVectorCube* keine Child-Widgets, sondern rendert das Layout durch den *CustomPainter*. Für die Darstellung der verschieden großen Registerzustände, sind vier Klas-

sen verantwortlich. Wie in Abschnitt 2.3 beschrieben ist die Diagrammart ab einer QuBit Anzahl von mehr als fünf unübersichtlich, weshalb auf eine Implementierung größerer Register verzichtet wird.

In Abbildung 4.5 ist eine Grafik zu sehen, die zum automatischen Testen des *CustomPainters* verwendet wird. Das Register befindet sich in einem Zustand komplexer Amplituden mit Überlagerung und Verschränkung.

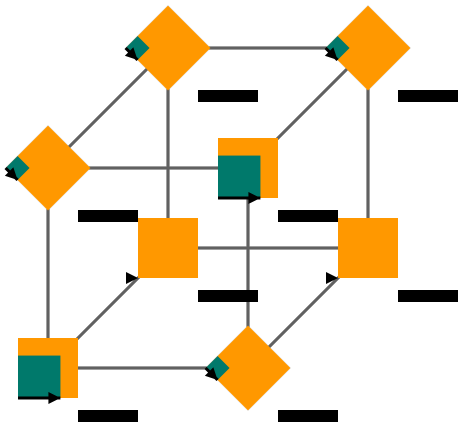


Abbildung 4.5.: *ThreeQBitCubePainter* golden Test Image

4.4. Darstellung der Übergänge

In Abbildung 4.6 ist zu sehen, wie sich CNOT auf einen beliebigen Zustand auswirkt, wenn das erste QuBit als Control und das zweite QuBit als Target eingesetzt ist. Wie in Abbildung 4.7 zu erkennen ist, wird bei Übergängen zunächst nach der Art des Übergangs unterschieden. Jede Klasse implementiert dann die Fälle für bis zu vier QuBits.

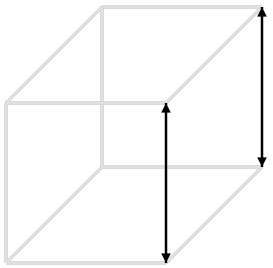


Abbildung 4.6.: *CXTransitionPainter* Darstellung der Anwendung eines CNOT Gates auf einem drei QuBit Registers

Die Pauli-X, -Y und -Z Gatter, sowie das Hadamard Gatter sind in der Darstellung ähnlich. Sie unterscheiden sich lediglich in der Art der Pfeilspitzen, nicht jedoch an der Position der Pfeile. Eine abstrakte Klasse *BasicGateTransition*, übernimmt dabei die Positionierung, die konkrete Klasse *XTransitionPainter* implementiert dabei lediglich einen *LineDrawer*.

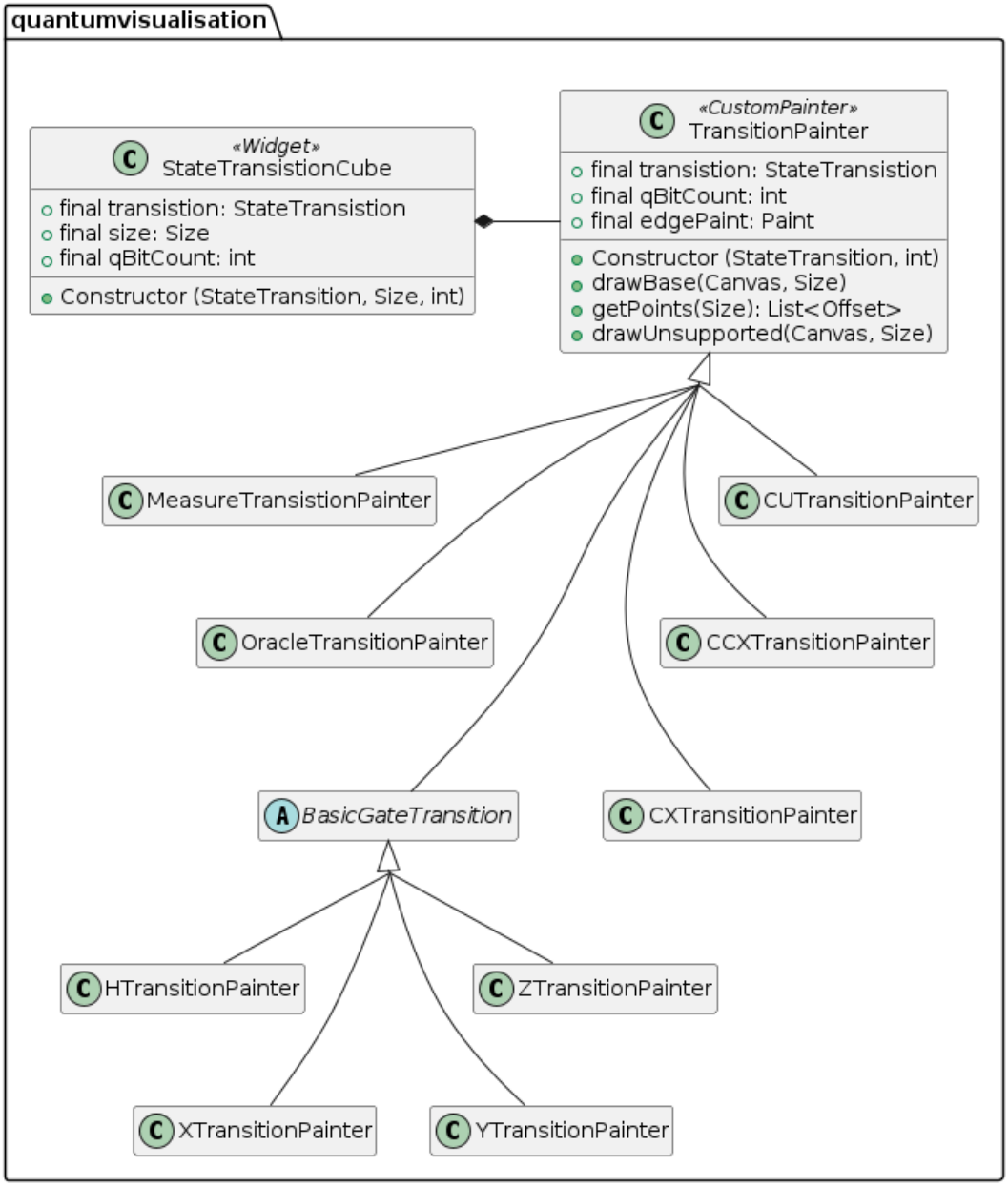


Abbildung 4.7.: UML Diagramm der Statevektor Transition Visualisierung

4.5. Statemanagement mit BLoC

In diesem Kapitel wird beschrieben, welche Statemanagement Optionen es gibt und welche Vorteile BLoC gegenüber den anderen präsentierten Optionen darstellt.

Ziel eines Statemanagement ist es systematisch Daten zu verarbeiten und die Darstellung dieser zu verwalten. Es gibt verschiedene Ansätze zum Statemanagement. Die einfachste Form Daten zu verwalten sind dabei *StatefullWidgets*, diese werden im Rahmen des Flutter-Frameworks mitgeliefert. Daten die in mehr als einem Widget benötigt werden müssen dann aufwendig weiter gegeben werden.

In komplexen Anwendungen wird meist eine von zwei großen Statemanagement Bibliotheken verwendet. Riverpod[3] ist eine dieser beiden Bibliotheken, BLoC ist die andere. Riverpod und BLoC unterscheiden sich in der API und sind in Funktion und Umfang etwa äquivalent. Die Entscheidung für BLoC beruht auf einer persönlichen Präferenz.

Das BLoC Pattern bietet eine Trennung von Datenhaltung und Darstellung. Abbildung 4.8 zeigt eine Vereinfachung des BLoC Patterns mit einem *cubit*, anstelle von Events können, aus den UI-Komponenten, Funktionen aufgerufen werden, die zum Beispiel das Laden von Daten aus einer Datei anstoßen. In der Anwendung die zu dieser Arbeit gehört, wird im *cubit* der Simulator aufgerufen. Wird vom Simulator ein Zwischenergebnis zurückgegeben, informiert der *cubit* alle UI-Komponenten, für die diese neuen Daten relevant sind.

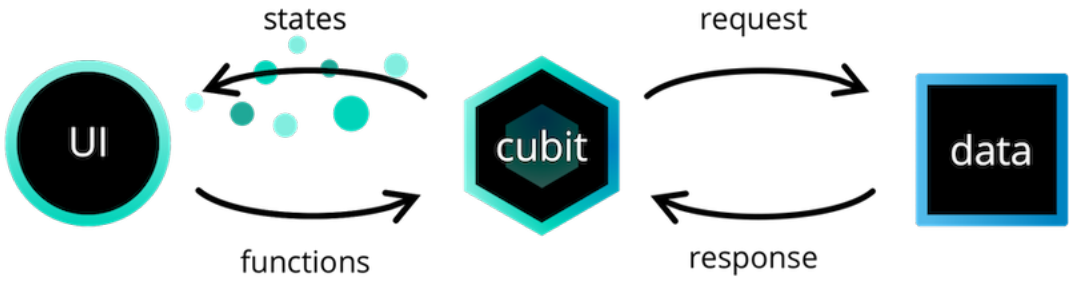


Abbildung 4.8.: Diagramm des BLoC Patterns mit Cubit[4]

Zusätzlich bietet die BLoC Bibliothek, eine Möglichkeit die korrekte Funktionsweise des Statemanagements systematisch zu Testen.

[3] *Riverpod*.
[4] Übernommen aus *Bloc State Management Library*

5. Phase Kickback nutzt Blackboxes

In den nächsten beiden Kapiteln wird ein Proof of Concept präsentiert. Es wird am Beispiel des Phase Kickback gezeigt, dass die Anwendung auch komplexere Szenarien des Quantencomputing im Würfeldiagramm visualisieren kann.

Phase Kickback ist ein wichtiges Prinzip des Quantencomputings. Grob gesprochen handelt es sich dabei um die Tatsache, dass Steuer-QuBits sich verändern, weil sie gesteuert haben. Wohingegen die Ziel-QuBits unverändert bleiben können. Im klassischen Computing kommt so etwas höchstens in Form von unerwünschten Seiteneffekten vor. Im Quantencomputing ist es ein wichtiges algorithmisches Prinzip.

Phase Kickback tritt in verschiedenen Szenarien auf. Dieses Kapitel soll einen Überblick darüber liefern, auf welche Weise Phase Kickback erreicht werden kann.

Abbildung 5.1 zeigt, dass Phase Kickback auf zwei Weisen erreicht werden kann. Zum einen mit Quantenorakeln, welche in Abschnitt 5.1 genauer beschrieben sind, zum anderen mit Controlled-U, dass in Abschnitt 5.2 erläutert wird.

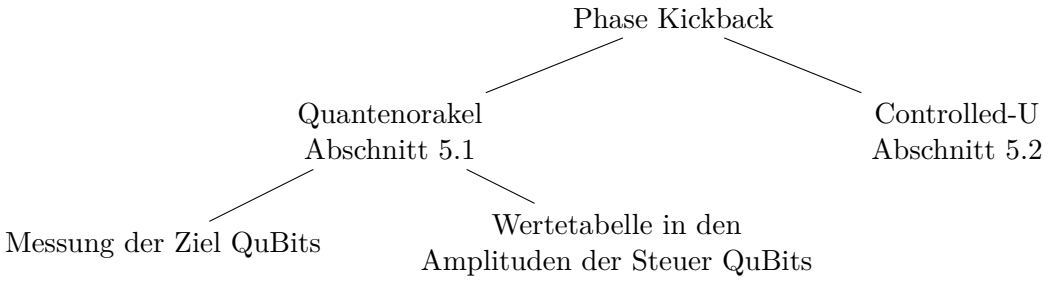


Abbildung 5.1.: Übersicht der verschiedenen Phase Kickback Szenarien[1]

Mit einem Quantenorakel lässt sich Phase Kickback auf zwei Arten verwenden. Zum einen kann nach einem Quantenorakel eine Messung auf den Ziel QuBits durchgeführt werden. Zum anderen ist die Wertetabelle, die dem Quantenorakel zugrunde liegt, in den Amplituden der Steuer QuBits enthalten. Beide Anwendungsfälle sind in Abschnitt 5.1 beschrieben.

[1] Vergleiche Just, *Kursmaterial Woche 1 | Einführung in Das Quantencomputing - Teil 3 |*

5.1. Blackbox „Quantenorakel“

Zu jeder klassischen booleschen Funktion $f(x_1, \ldots, x_n) = y$ lässt sich ein Quantenschaltkreis aufbauen, der aus den Quantengattern Pauli-X, CNOT und Toffoli besteht, und die Funktion f „berechnet“. Das bedeutet, dass der Quantenschaltkreis $n + 1$ QuBits bedient.

Wird der Quantenschaltkreis auf einen Basiszustand $|x_0, x_1, \ldots, x_{n-1}, y\rangle$ angewendet, liefert er den Basiszustand $|x_0, x_1, \ldots, x_{n-1}, (y \oplus f(x_0, x_1, \ldots, x_{n-1}))\rangle$. So einen Quantenschaltkreis nennt man Quantenorakel für die Funktion f. Ist der Input für den Quantenschaltkreis kein Basiszustand, sondern eine Superposition von Basiszuständen, befindet sich das Register am Ende in Superposition der Inputs und Outputs.

In Abbildung 5.2 ist zu sehen, wie eine boolesche Funktion verwendet werden kann. Die QuBits $|x_1\rangle$ bis $|x_n\rangle$ sollen dabei den Input Bits der Funktion f entsprechen, das unterste QuBit repräsentiert den Output. In Anhang III findet sich eine Auflistung aller booleschen Funktionen mit zwei Input-Bits. Diese lassen sich beliebig kombinieren, um boolesche Funktionen mit drei oder mehr Inputs aufzubauen. Dafür sind unter Umständen zusätzliche QuBits für Zwischenergebnisse notwendig.

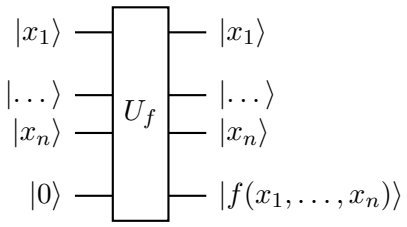


Abbildung 5.2.: Auswirkungen von U_f auf Ergebnis-QuBit

Ein Quantenorakel für eine Funktionen $f : \{0,1\}^n \to \{0,1\}$ ist ein Quantenschaltkreis, bei dem für alle Basiszustände $|x_1 x_2 \ldots x_n y\rangle$ gilt:

> Nach dem Quantenorakel befinden sich die Input QuBits im gleichen Zustand, wie vor dem Orakel, das Output QuBit $|y\rangle$ befindet sich im Zustand $|y \oplus f(x_1, \ldots, x_n)\rangle$.

In Abbildung 5.3 ist diese Voraussetzung veranschaulicht.

Wie ein Quantenorakel aufgebaut werden, kann ist in Just, *Kursmaterial Woche 1 | Einführung in Das Quantencomputing - Teil 3 |*, Video 3 beschrieben. Eine genauere Erläuterung geht über den Rahmen dieser Arbeit hinaus.

Eine Matrix für ein Orakel zur Funktion $f(x_1, x_2) = (\neg x_1) \wedge x_2$, die ohne zusätzliche QuBits auskommt, hat die Form wie in Gleichung 5.1.

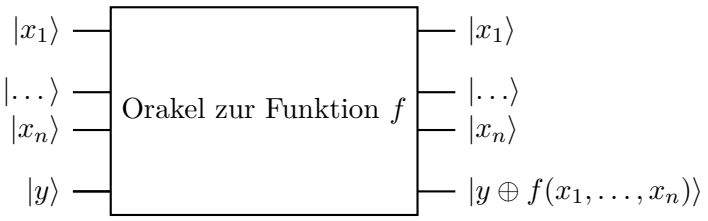


Abbildung 5.3.: Auswirkungen von U_f auf Ergebnis QuBit

$$M = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{pmatrix} \tag{5.1}$$

Auf der Diagonalen liegen 2×2 Matrizen, entweder M_{id} oder M_{not}, abhängig davon, ob die Funktion f an der Stelle 0 oder 1 ist.

$$M_{id} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad M_{not} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{5.2}$$

Ein Vorteil ergibt sich aus einem Quantenorakel, wenn sich U_f als Schaltkreis beschreiben lässt, ohne das dafür die gesamte Wertetabelle aufgestellt werden muss. Wenn der Quantenschaltkreis „klein“ ist, lässt sich in diesem Fall die Funktion für einzelne Inputs berechnen. Verwendet man als Input eine Superposition von Quantenzuständen, führt dies dazu, dass im Prinzip die komplette Wertetabelle berechnet wird. Der zielgenaue Zugriff auf einzelne Outputs durch Messung ist dabei jedoch nicht möglich.

Das erste Szenario für Phase Kickback verwendet Quantenorakel mit anschließender Messung der Ziel QuBits. Für diese Form wird der Schaltkreis so vorbereitet, dass sich die Input QuBits in einem gleichmäßigen überlagerten Zustand befinden. Mit positiven Amplituden an allen Stellen, bei denen das Ziel QuBit $|0\rangle$ ist. Abbildung 5.4 zeigt den Zustand des Registers, bevor das Quantenorakel angewendet wird.

Das Quantenorakel vertauscht entlang der Achse des Ziel QuBit die Amplituden. Abbildung 5.5 zeigt die Wirkung des Orakels zur Funktion

$$f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$$

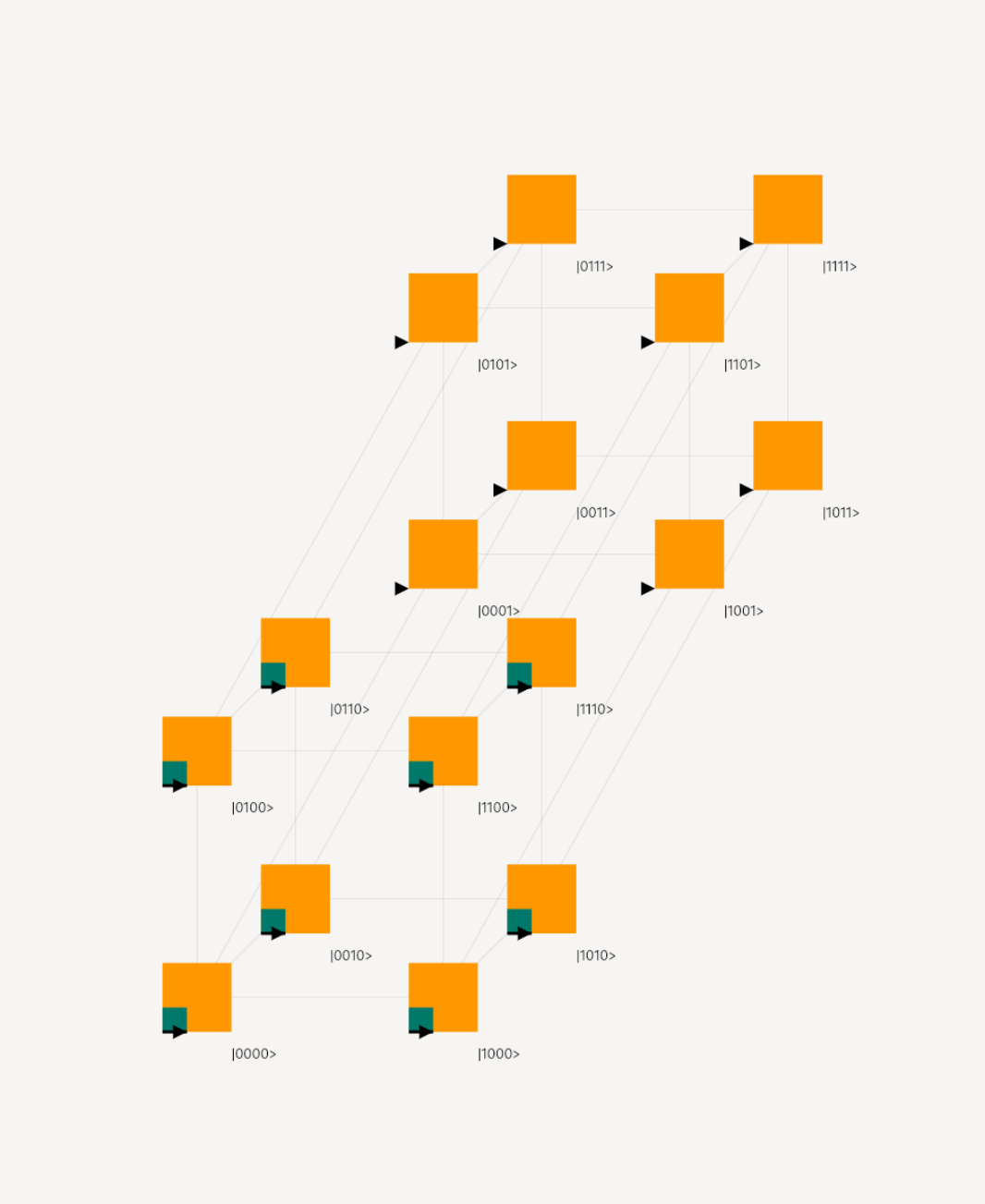


Abbildung 5.4.: Registerzustand vor dem Quantenorakel

.

Nach dem Quantenorakel liegen dann ein Teil der Amplituden im vorderen Würfel, an den Ecken in denen $f(x_1, x_2, x_3) = 0$ ist. Im hinteren Würfel liegen die Amplituden in den Ecken für die $f(x_1, x_2, x_3) = 1$ ist.

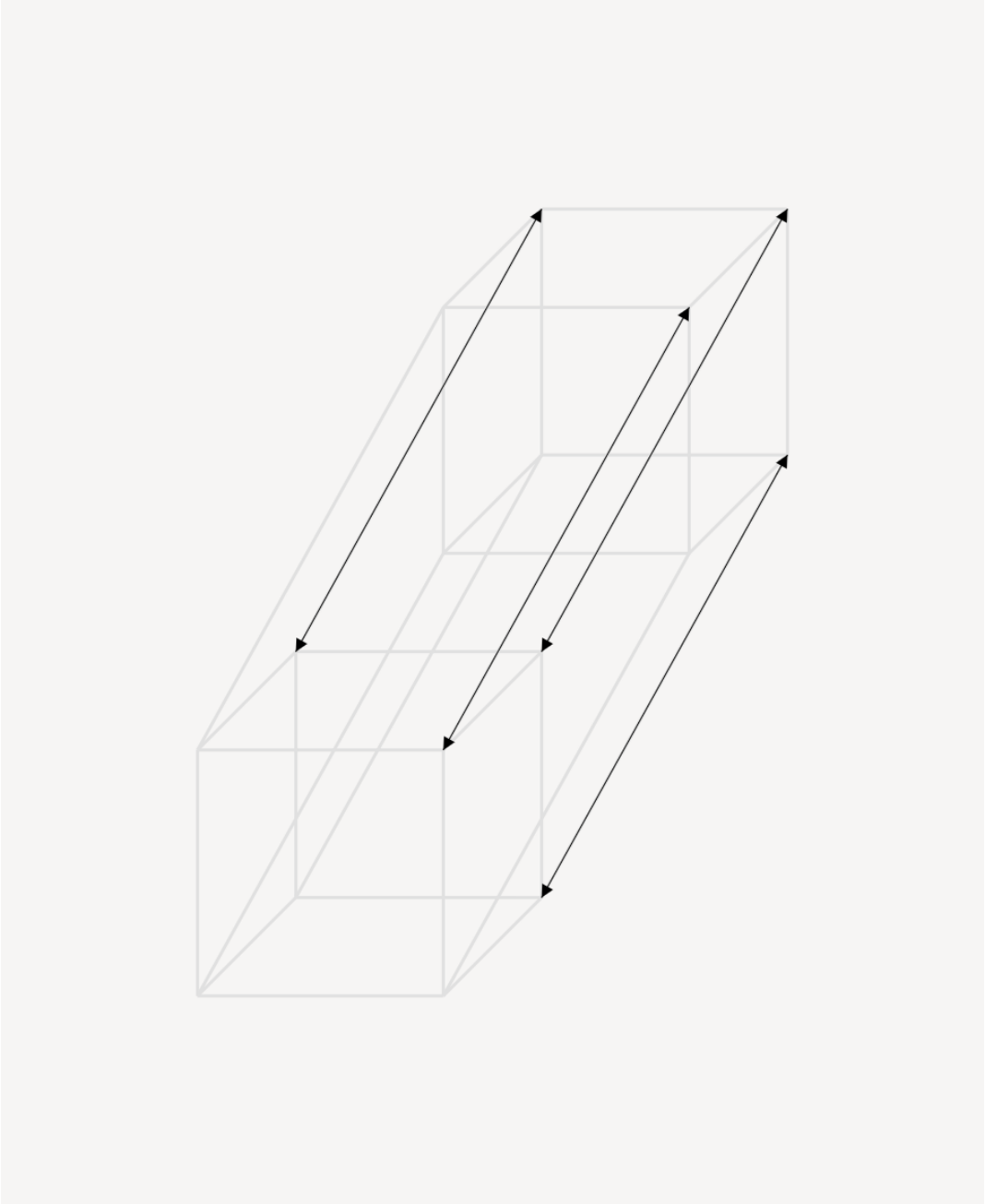


Abbildung 5.5.: Registerzustand vor dem Quantenorakel für Szenario 1

Wird nach dem Quantenorakel das Ziel QuBit gemessen bleiben nur die Amplituden übrig, bei denen die Funktion f den gleichen Wert als Ergebnis liefert, der am Ziel QuBit gemessen ist. Abbildung 5.6 zeigt die beiden möglichen Zustände nach der Messung des Ziel QuBit.

Im zweiten Szenario wird die Wertetabelle in den Amplituden der Steuer-QuBits co-

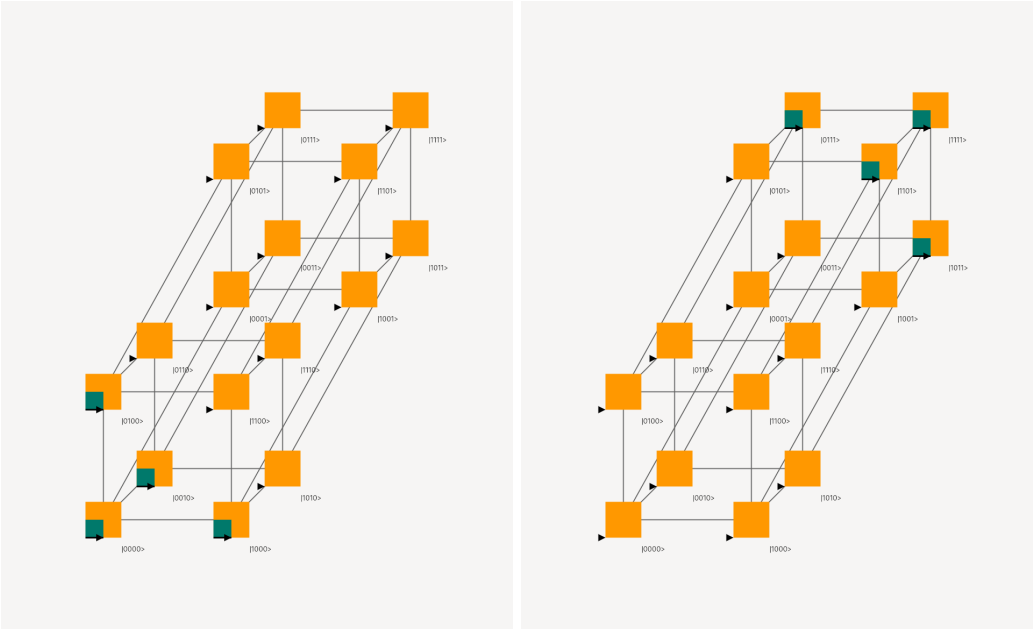


Abbildung 5.6.: Registerzustand nach der Messung (links 0, rechts 1)

diert. Dafür wird das Register wieder vorbereitet, diesmal mit einem gleichmäßigen überlagerten Zustand, mit positiven Amplituden, wo das Ziel QuBit 0 ist und negativen Amplituden, wo das Ziel QuBit 1 ist. Abbildung 5.7 zeigt den vorbereiteten Registerzustand vor Anwendung des Orakels.

Als Funktion für das Quantenorakel wird die gleiche Funktion $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$ verwendet, die im ersten Szenario bereits beschrieben ist. Dementsprechend ist die Wirkung auf das Register wieder in Abbildung 5.5 zu sehen.

Nach dem Quantenorakel befindet sich das Register im Zustand aus Abbildung 5.8. Zu sehen ist, dass die Vorzeichen der Amplituden an den Stellen getauscht sind, an denen $f = 1$ ist.

Mit einer geschickten Nachbereitung kann dann eine globale Eigenschaft der Funktion f ermittelt werden, ohne das dafür jeder Input ausprobiert werden muss. Bekannte Beispiele für Algorithmen, welche auf diese Weise arbeiten, ist der Algorithmus von Deutsch, beziehungsweise der Algorithmus von Deutsch-Jozsa, welche als Nachbereitung des Registers die Hadamard Transformation verwenden.

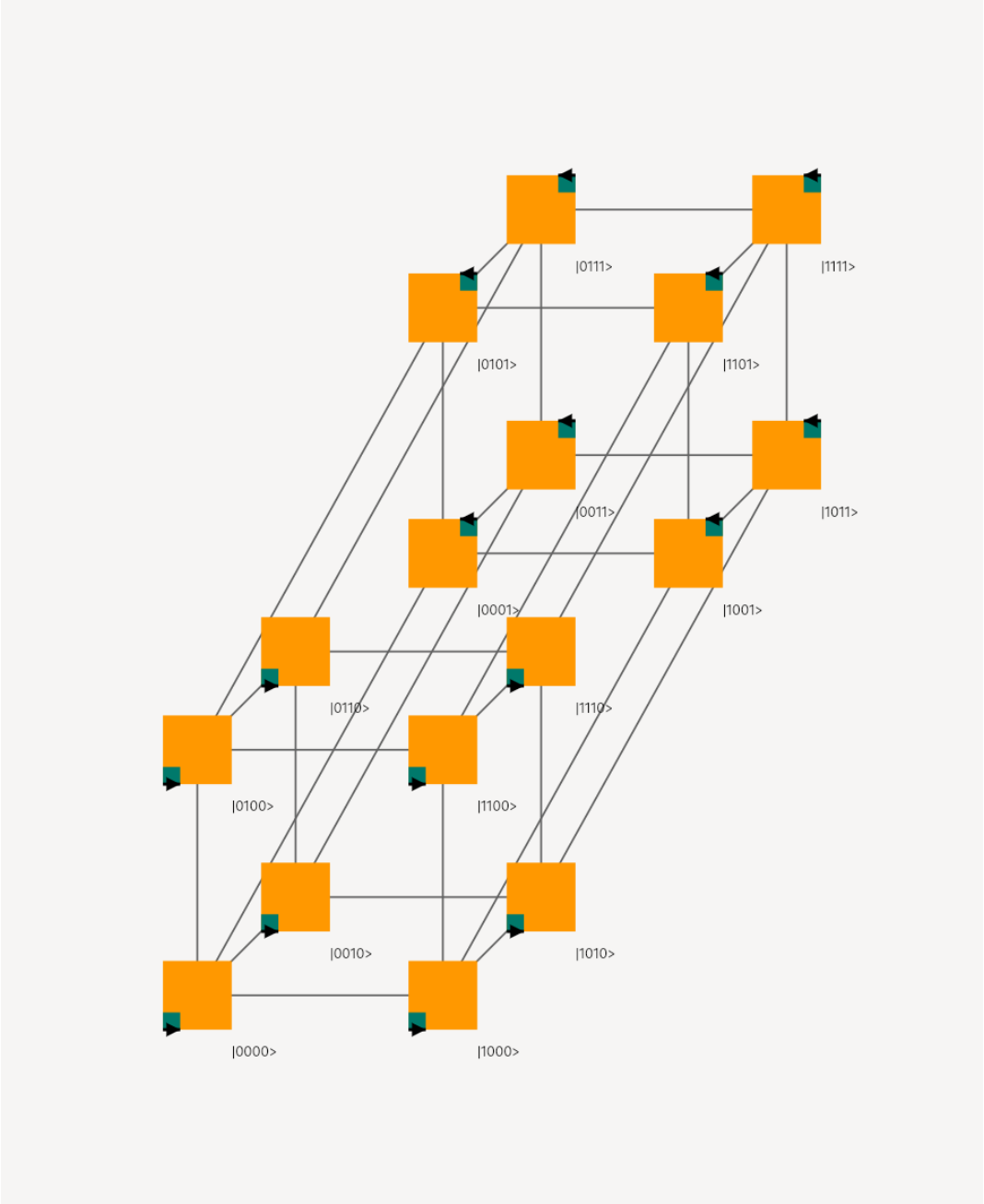


Abbildung 5.7.: Registerzustand vor dem Quantenorakel für Szenario 2

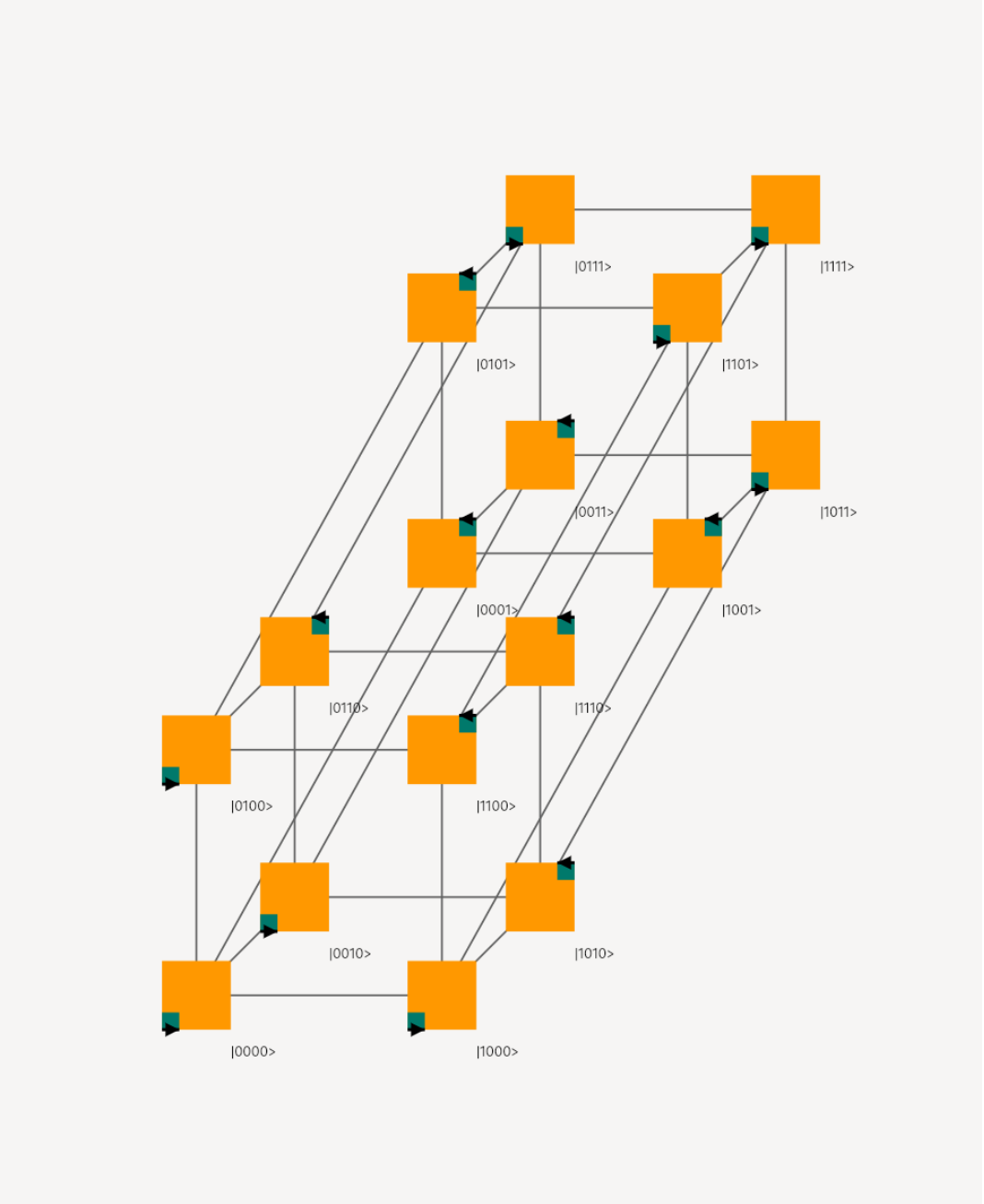


Abbildung 5.8.: Registerzustand nach dem Quantenorakel für Szenario 2

5.2. Blackbox „Controlled-U“

In diesem Abschnitt wird Phase Kickback mit Controlled-U Gattern beschrieben. Dabei ist U eine Transformation die auf n QuBits ausgeführt wird. Controlled-U nutzt ein zusätzliches Steuer-QuBit. Ein Beispiel für so ein Controlled-U Gatter, ist CNOT, mit dem Pauli-X Gatter als Transformation U. Die genaue Wirkung von Controlled-U wird im Laufe dieses Abschnitts dargestellt. Dabei wird den Darstellungen aus Just, *Kursmaterial Woche 1 | Einführung in Das Quantencomputing - Teil 3 |* gefolgt.

Betrachtet wird zunächst nur eine einzelne Ecke eines Würfeldiagramms. Abbildung 5.9 zeigt eine Drehung der Amplitude um 90°. Gleichung 5.3 beschreibt den Zustand vor der Drehung, die Gleichung 5.4 den Zustand danach.

$$\alpha_{\text{vor}} = \sqrt{\frac{1}{4}} = \frac{1}{2} \cdot e^{i \cdot 0°} \tag{5.3}$$

$$\alpha_{\text{nach}} = \sqrt{\frac{1}{4}} i = \frac{1}{2} \cdot e^{i \cdot 90°} \tag{5.4}$$

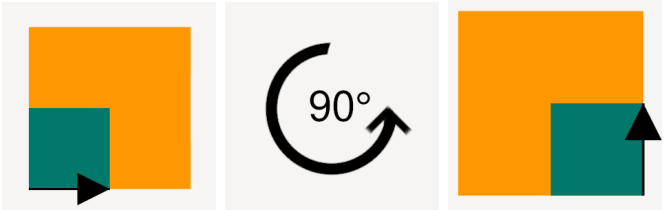


Abbildung 5.9.: Drehung einer Amplitude um 90°[2]

Genau wie eine einzelne Amplitude gedreht werden kann, lassen sich auch alle Amplituden eines Registers, um einen Winkel drehen. Abbildung 5.10 zeigt wie eine Rotation aller Amplituden eines zwei QuBit Registers, um den Winkel 30° gedreht werden.

Im Folgenden wird betrachtet, welche Auswirkungen das Pauli-X Gatter auf verschied Zustände hat. Gleichung 5.5 zeigt den Zustand, vor Pauli-X, die Gleichung 5.6 danach.

$$|\psi_{\text{vor}}\rangle = \sqrt{\frac{1}{4}} |0\rangle - \sqrt{\frac{3}{4}} |1\rangle \tag{5.5}$$

$$|\psi_{\text{nach}}\rangle = -\sqrt{\frac{3}{4}} |0\rangle + \sqrt{\frac{1}{4}} |1\rangle \tag{5.6}$$

Wie in Abbildung 5.11 zu sehen ist vertauscht Pauli-X die Amplituden, diese Vertauschung ist im allgemeinen Fall keine Rotation.

[2] Vergleiche Just, *Kursmaterial Woche 1 | Einführung in Das Quantencomputing - Teil 3 |* Video 8
[3] Vergleiche Just, *Kursmaterial Woche 1 | Einführung in Das Quantencomputing - Teil 3 |* Video 8

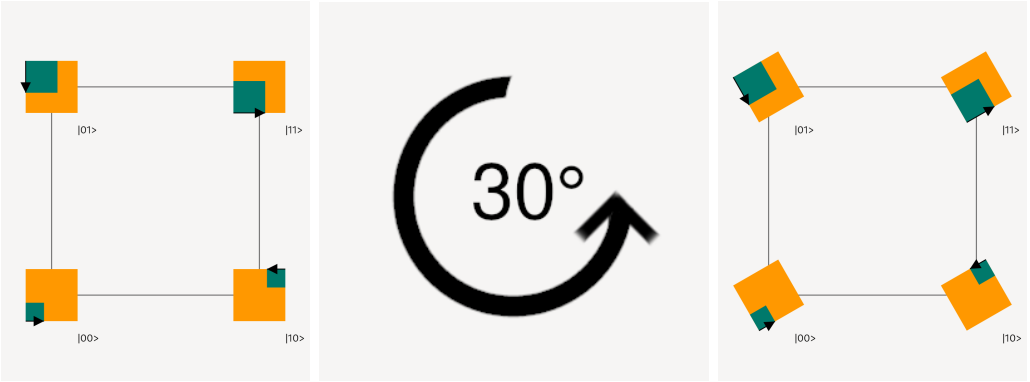


Abbildung 5.10.: Drehung aller Amplituden eines Registers um 30°[3]

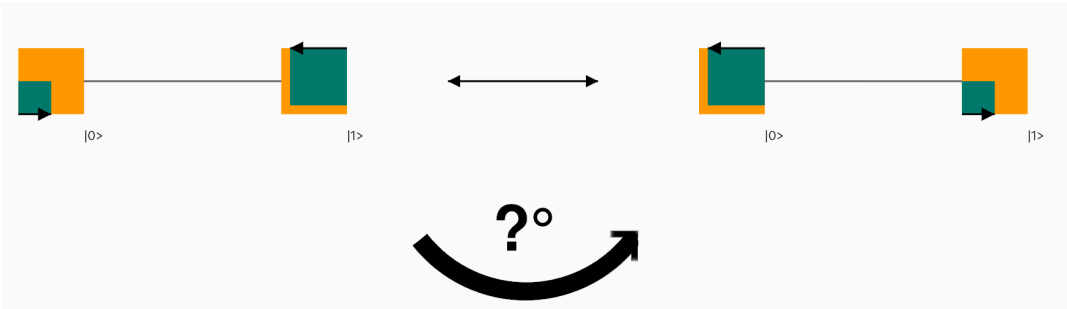


Abbildung 5.11.: Auswirkungen von Pauli-X auf Zustand aus Gleichung 5.5[4]

Wird das Register mit einem Eigenzustand der Pauli-X Matrix initialisiert, kann die Auswirkung als Rotation angesehen werden. Der Wert $\sqrt{1/2}\,|0\rangle - \sqrt{1/2}\,|1\rangle$ aus Gleichung 5.7 ist ein Eigenzustand für Pauli-X zum Eigenwert -1. Der Eigenwert muss dabei in jedem Fall den Betrag 1 haben, denn der Betrag des Zustandsvektors muss sowohl vor als auch nach dem Quantengatter gleich 1 sein. Damit entspricht Pauli-X auf diesem Zustand einer Rotation um 180°. Veranschaulicht ist diese Rotation in Abbildung 5.12.

$$|\psi_{\text{eigen}}\rangle = \sqrt{\frac{1}{2}}\,|0\rangle - \sqrt{\frac{1}{2}}\,|1\rangle \tag{5.7}$$

Anstelle des Pauli-X Gatters wird jetzt ein Schaltkreis als Quantengatter betrachtet. Die Matrix, die die Funktionsweise des Quantschaltkreises aus Abbildung 5.13 beschreibt, wird mit U bezeichnet. In Abbildung 5.14 wird die Auswirkungen von U auf einen Zustand $|\psi\rangle$ beispielhaft dargestellt.

Abbildung 5.15 zeigt, dass U auf dem Eigenzustand $\sqrt{1/2}\,|0\rangle - \sqrt{1/2}\,|1\rangle$ einer Rotation um 90° entspricht. Der Eigenwert zu diesem Eigenzustand ist i. Eine Multiplikation mit einer komplexen Zahl kann geometrisch betrachtet werden. Dabei wird

[4]Vergleiche Just, *Kursmaterial Woche 1 | Einführung in Das Quantencomputing - Teil 3* | Video 8
[5]Vergleiche Just, *Kursmaterial Woche 1 | Einführung in Das Quantencomputing - Teil 3* | Video 8

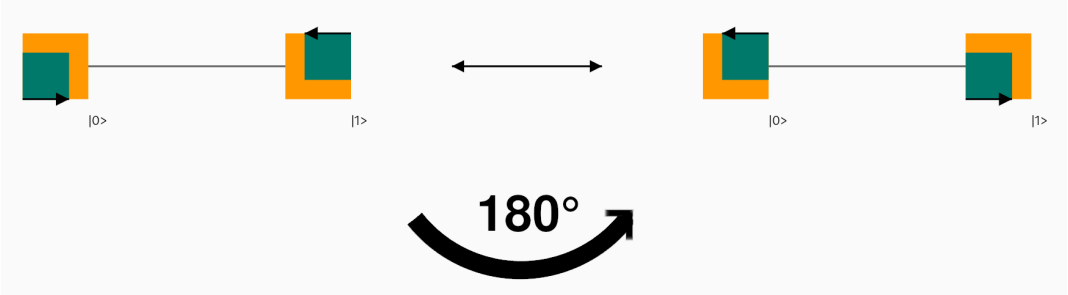


Abbildung 5.12.: Auswirkungen von Pauli-X auf Zustand aus Gleichung 5.7[5]

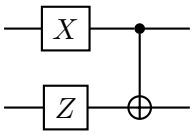


Abbildung 5.13.: Schaltkreis für ein beispielhaftes U-Gatter

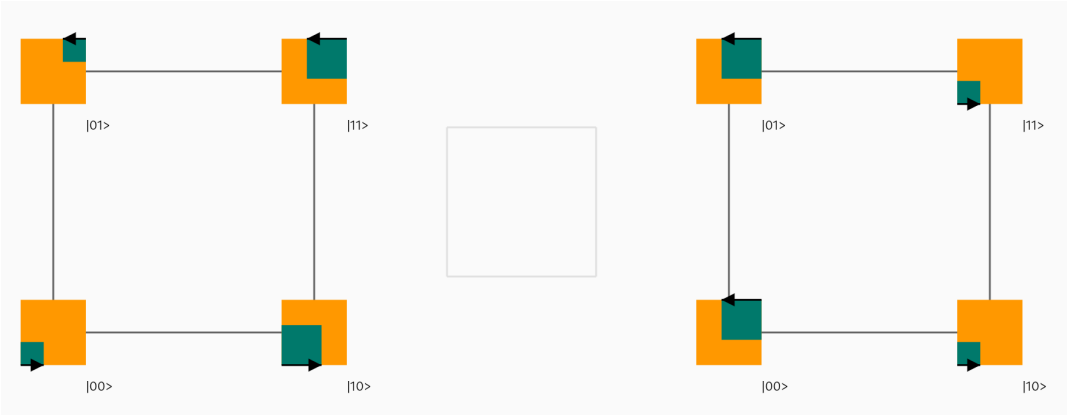


Abbildung 5.14.: Auswirkungen von U auf einen beliebigen Zustand ψ

die komplexe Zahl jeweils als Vektor in einem Polarkoordinatensystem betrachtet. Die Längen der Vektoren werden dann multipliziert und die Winkel gegenüber der x-Achse addiert.

Betrachtet man die unitäre Matrix eines Quantengatters, so lässt sich eine Matrix aufbauen, die einer Controlled-Version dieses Quantengatters entspricht. Ein Beispiel für so ein Controlled-U Gatter ist CNOT, CNOT liegt das Pauli-X Gatter zugrunde. Betrachtet man die Matrizen in Gleichung 5.8 und 5.9 fällt auf, dass im unteren

[6]Vergleiche Just, *Kursmaterial Woche 1 | Einführung in Das Quantencomputing - Teil 3* | Video 8

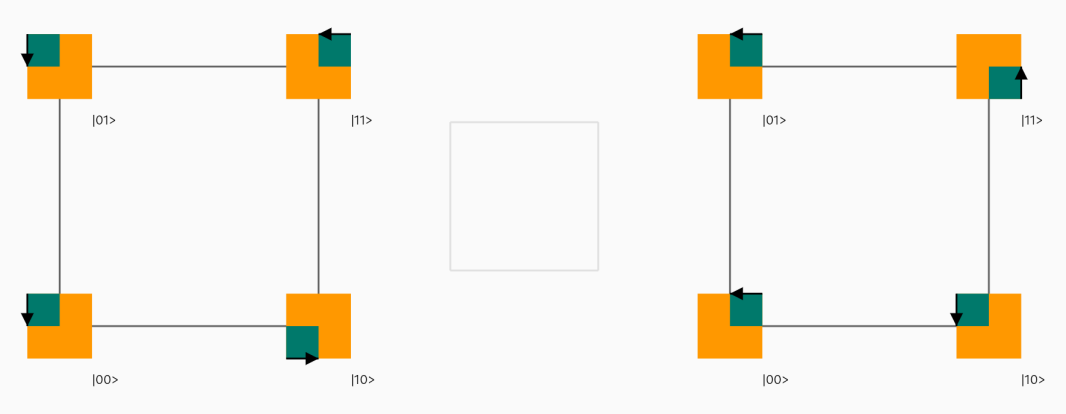


Abbildung 5.15.: Auswirkungen von U auf einen Eigenzustand von U^6

rechten Quadranten von CNOT die Pauli-X Matrix enthalten ist.

$$M_X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{5.8}$$

$$M_{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{5.9}$$

Nach diesem Prinzip lassen sich für alle Quantengatter auch Controlled-Versionen erstellen. Für eine Matrix der Größe $2^n \times 2^n$ ergibt sich eine Controlled Matrix der Größe $2^{n+1} \times 2^{n+1}$. Diese Matrix lässt sich in vier Quadranten, wie in Gleichung 5.10 aufteilen. Die Matrix behandelt das erste QuBit als Steuerbit und wendet U auf die folgenden QuBits entsprechend der Größe von U genau da an, wo das erste QuBit gleich 1 ist.

$$M_{cu} = \begin{pmatrix} Id_{2^n} & 0 \\ 0 & U \end{pmatrix} \tag{5.10}$$

Wichtig ist, dass die Größe der Matrix Id_{2^n}, sowie die Größe der 0 Matrizen gleich der Größe von U ist. Abbildung 5.16 zeigt, wie sich das Controlled-U Gatter auf einen beliebigen Zustand auswirkt. Auf der Seite des Würfels, bei der das Steuer-QuBit 0 ist ändert sich an den Amplituden nichts, auf der anderen Seite, wo das Steuer-QuBit 1 ist, wird das Gatter U angewendet.

In Abbildung 5.17 sind zwei Register zu sehen. Eines auf einem QuBit im zufälligen Zustand $|q\rangle$, das andere ist ein zwei QuBit Register in einem Eigenzustand für das Gatter U zum Eigenwert i. Wie in Abschnitt 2.2 beschrieben, können beide Register

[7] Vergleiche Just, *Kursmaterial Woche 1 | Einführung in Das Quantencomputing - Teil 3 |*

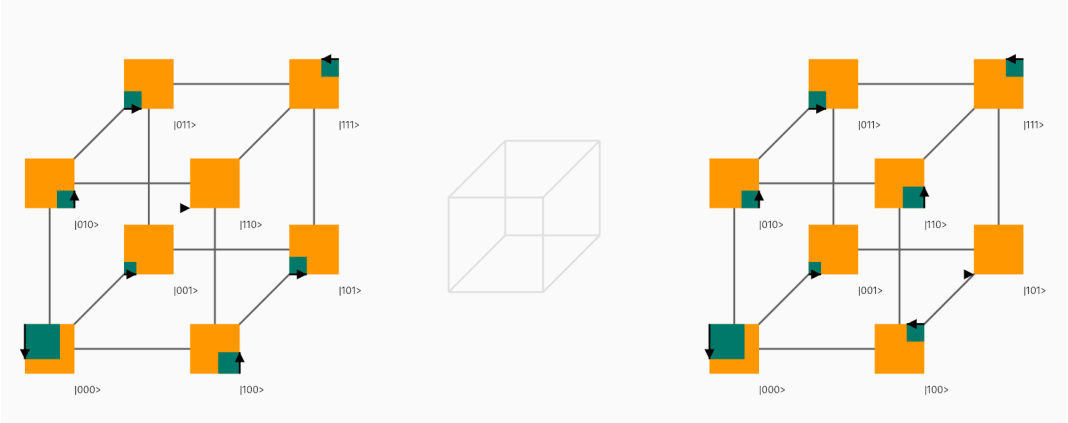


Abbildung 5.16.: Auswirkungen von Controlled-U auf einen beliebigen Zustand[7]

auch als ein gemeinsames Register betrachtet werden. Eine Visualisierung des Registers aus allen drei QuBits ist in Abbildung 5.18 zu sehen. Geometrisch betrachtet sind die Amplituden auf der rechten Seite ein Vielfaches derer auf der linken Seite.

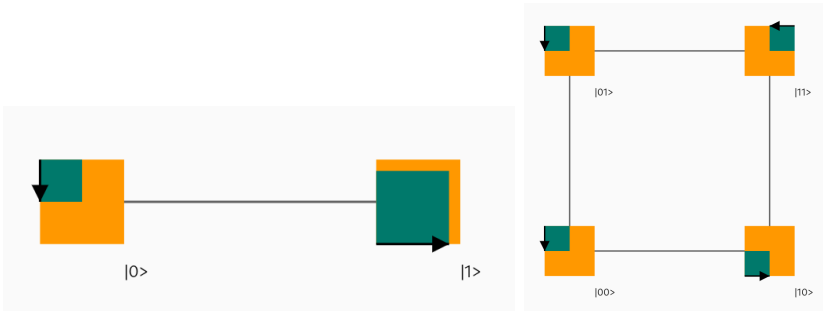


Abbildung 5.17.: Würfeldiagramm für $|q\rangle$ links, $|\psi_{eigen}\rangle$ rechts

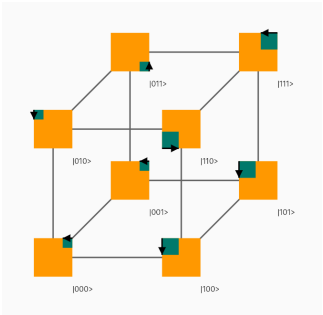


Abbildung 5.18.: Würfeldiagramm für $|q\rangle \otimes |\psi_{eigen}\rangle$

Gleichung 5.11 beschreibt den Zustand, bevor das Quantenorakel ausgeführt ist. Man beachte, dass es sich bei $|\psi\rangle$ nicht um ein einzelnes QuBit handelt. Wird Controlled-U auf diesem Zustand ausgeführt, hat dies die Wirkung von U auf dem Teil an

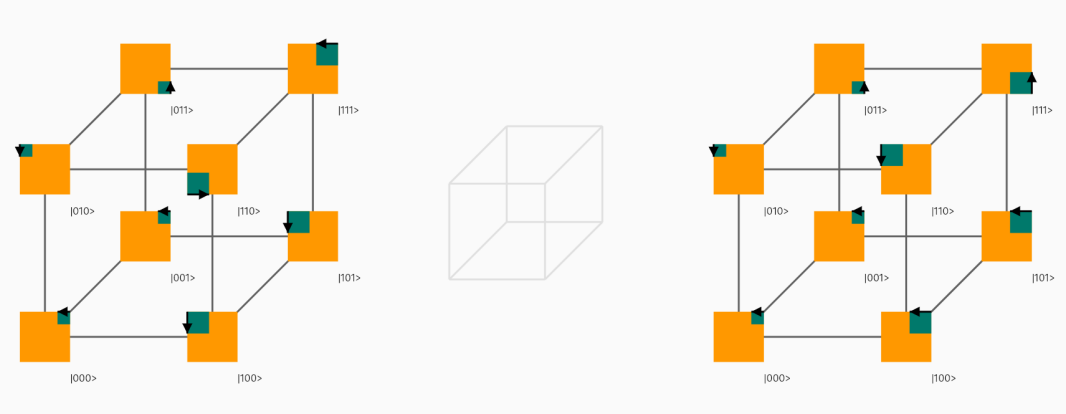


Abbildung 5.19.: Auswirkungen von Controlled-U auf $|q\rangle \otimes |\psi_{eigen}\rangle$

dem das Steuer-QuBit 1 ist. Gleichung 5.12 beschreibt diese Wirkung, da sich $|\psi\rangle$ in einem Eigenzustand von U befindet, ergibt sich Gleichung 5.13. Grafisch ist dies in Abbildung 5.19 dargestellt.

In Gleichung 5.13 steht $|\psi\rangle$ sowohl bei $|0\rangle$ als auch bei $|1\rangle$ mit Ausnahme eines Vorfaktors unverändert. Somit sind die QuBits unverschränkt. In Gleichung 5.14 wird $|\psi\rangle$ ausgeklammert. λ bleibt als Rotation der Amplitude am Steuer-QuBit zurück. Teilt man das Register wieder auf, erhält man das Würfeldiagramm aus Abbildung 5.20.

$$(q_0 |0\rangle + q_1 |1\rangle) \otimes |\psi\rangle = q_0 |0\rangle \otimes |\psi\rangle + q_1 |1\rangle \otimes |\psi\rangle \tag{5.11}$$

$$q_0 |0\rangle \otimes |\psi\rangle + q_1 |1\rangle \otimes U |\psi\rangle \tag{5.12}$$
$$= q_0 |0\rangle \otimes |\psi\rangle + q_1 |1\rangle \otimes \lambda |\psi\rangle \tag{5.13}$$
$$= (q_0 |0\rangle + \lambda q_1 |1\rangle) \otimes |\psi\rangle \tag{5.14}$$

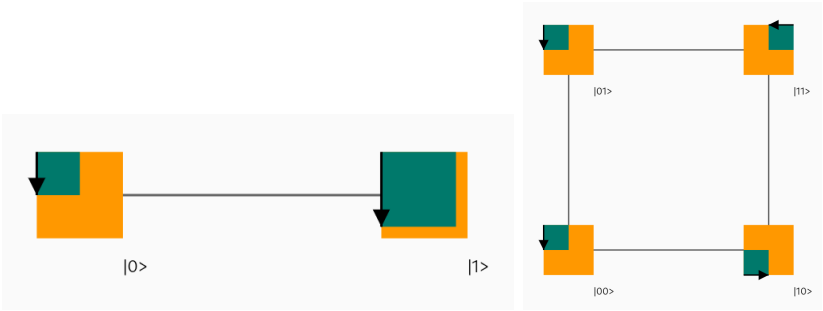


Abbildung 5.20.: Würfeldiagramm für $q_0 |0\rangle + \lambda q_1 |1\rangle$ links, $|\psi_{eigen}\rangle$ rechts

6. Proof of Concept für Phase Kickback

In diesem Kapitel wird betrachtet, ob die *QuantenLernApp* auch für Schaltkreise verwendet werden kann, die mit Phase Kickback arbeiten.

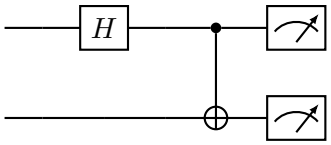


Abbildung 6.1.: EPR-Paar Schaltkreis

In Abbildung 6.1 ist ein Quantenschaltkreis zum sehen, der ein EPR-Paar erstellt und dann misst. Wie dieser Quantenschaltkreis als Datenstruktur angelegt werden kann, ist in Listing 6.1 zu lesen. Die Verwendung des initialen Vektors ist in Abschnitt 6.1 genauer beschrieben.

Listing 6.1: EPR-Paar als Schaltkreis aufbauen

```
import 'package:quantumsimulator/quantumsimulator.dart';

var circuit = Circuit(
  initial: initial, //Initialer Vektor
  steps: [
    H(0),
    CX(0, 1),
    MeasureStep(index: 0),
    MeasureStep(index: 1),
  ],
);
```

Die Anzahl der QuBits wird aus der Größe des initialen Vektors errechnet. Vorausgesetzt der Schaltkreis hat genug QuBits, können beliebige Quantengatter angewendet werden.

6.1. Komplexe Initialwerte für QuBits

Der Konstruktor der *Circuit* Klasse fordert einen Initialvektor, der den Zustand des Registers vor dem ersten Berechnungsschritt beschreibt. Um valide zu sein, muss ein Vektor als eine Bedingung die Größe 2^n haben, wobei n die Anzahl der QuBits ist.

In Listing 6.2 ist zu sehen, wie ein Schaltkreis mit dem initialen Zustand $|00\rangle$ instanziiert werden kann.

Listing 6.2: Schaltkreis mit Vektor initialisieren

```
import 'package:quantumsimulator/quantumsimulator.dart';

var circuit = Circuit(
  initial: Vector([
    ComplexDouble.one(),
    ComplexDouble(),
    ComplexDouble(),
    ComplexDouble(),
  ]),
  steps: [
    ...
  ],
);
```

Eine weitere Voraussetzung für die Validität des Initialvektors ist, dass die Summe der Quadrate der Elemente gleich 1 ist, so wie in Abschnitt 2.2 beschrieben.

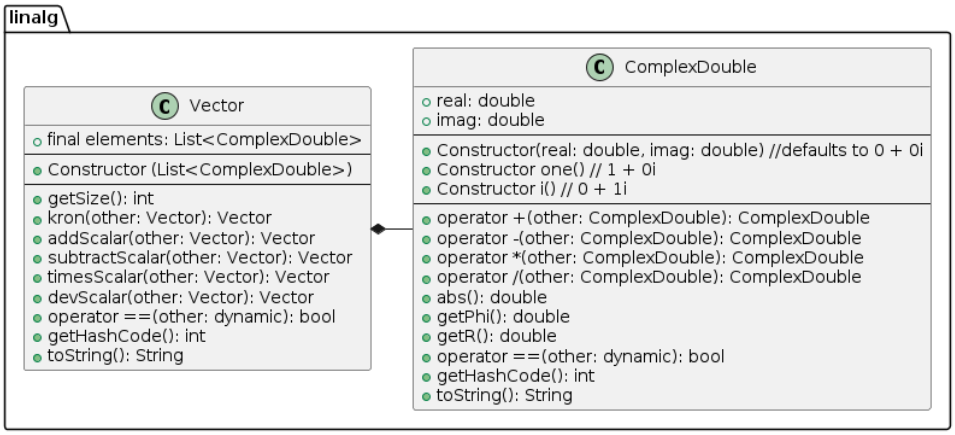


Abbildung 6.2.: UML Diagramm Ausschnitt aus *linalg* Paket

Wie in Abbildung 6.2 zu sehen ist, ist die *Vector*-Klasse für komplexe Zahlen implementiert. Damit ist die Voraussetzung, dass Schaltkreise mit komplexen Initialwerten initialisiert werden können, erfüllt.

6.2. Simulation und Visualisierung von „Quantenorakeln“

Wie in Abschnitt 5.1 beschrieben, liegt einem Quantenorakel eine boolesche Funktion zugrunde. Der Konstruktor der *Oracle*-Klasse nimmt eine solche Funktion entgegen und errechnet, mit dem in Listing 6.3 beschriebenen Algorithmus, die Matrix des

Quantengatters. So kann ein Quantenorakel im Simulator auf die gleiche Weise verwendet werden, wie die anderen Quantengatter, die eine bestimmte Matrix haben.

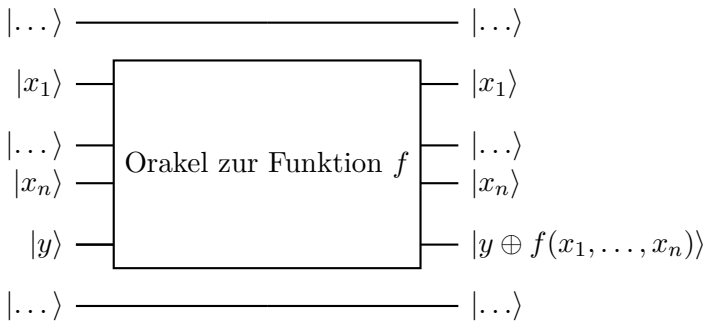


Abbildung 6.3.: Schaltkreis mit Quantenorakel

Listing 6.3: Aufbauen einer Matrix für ein Quantenorakel zur Funktion f

```
List<List<double>> m = [];
for (var i = 0; i < pow(2, numInputQBits).round(); i++) {
    //generate one line of the diagonal matrix (linenumber mod 2 == 0)
    var v1 = List.generate(
      pow(2, numInputQBits + 1).round(),
      (index) => index == 2 * i ? 1.0 : 0.0,
    );
    //generate one line of the diagonal matrix (linenumber mod 2 == 1)
    var v2 = List.generate(
      pow(2, numInputQBits + 1).round(),
      (index) => index == 2 * i + 1 ? 1.0 : 0.0,
    );
    //evaluate if f(given inputs is true)
    var output = f(inputs[i]);
    if (output) {
    // add lines like Pauli-X Gate
      m.add(v2);
      m.add(v1);
    } else {
    // add lines like ID Gate
      m.add(v1);
      m.add(v2);
    }
}
return Matrix.ofDouble(m);
```

Eine Einschränkung bei der Implementierung ist, dass ein Quantenorakel nicht auf beliebige QuBits angewendet werden kann. Nur die Konstellation aus Abbildung 6.3 ist möglich. Die Input QuBits müssen in richtiger Reihenfolge und aufeinander folgend liegen und das Output QuBit muss nach dem letzten Input QuBit folgen, ohne dass unbeteiligte QuBits dazwischen liegen. Eine Verallgemeinerung der Implementierung ist in Zukunft denkbar, überschreitet jedoch den Umfang dieser Arbeit.

Für die Visualisierung der Registerzustände, vor und nach Anwenden des Quantenorakels, wird keine zusätzliche Information benötigt. Wie bei den anderen Quantengattern enthält der Vektor die notwendigen Informationen, um als Würfel dargestellt zu werden.

Der Übergang von einem Registerzustand in den nächsten ist etwas komplizierter. Für einfache Quantengatter wie Pauli-X ist lediglich relevant, auf welchem QuBit das Quantengatter angewendet wird. Bei einem Quantenorakel ist zusätzliche wichtig, für welche Input Werte die Funktion f wahr (1) ist. Diese Information wird jeweils in einer Instanz der *StateTransition* Klasse bereitgestellt. Die *StateTransition* wird dabei schon im Konstruktor des jeweiligen Quantengatters angelegt, der Simulator gibt diese zusammen mit dem berechneten Ergebnis als *SimulationResult* weiter. Listing 6.4 zeigt wie eine *StateTransition* für ein Quantenorakel aufgebaut ist.

Listing 6.4: Aufbauen der *StateTransition* für ein Quantenorakel

```
StateTransition(
  TransitionType.ORACLE,
  [
    index, // index of the first affected qbit
    numInputQBits, // number of input qbits for the oracle
    ...ones, // all the cases that f is true,
             //integers in binary format represent a list of booleans
  ],
);
```

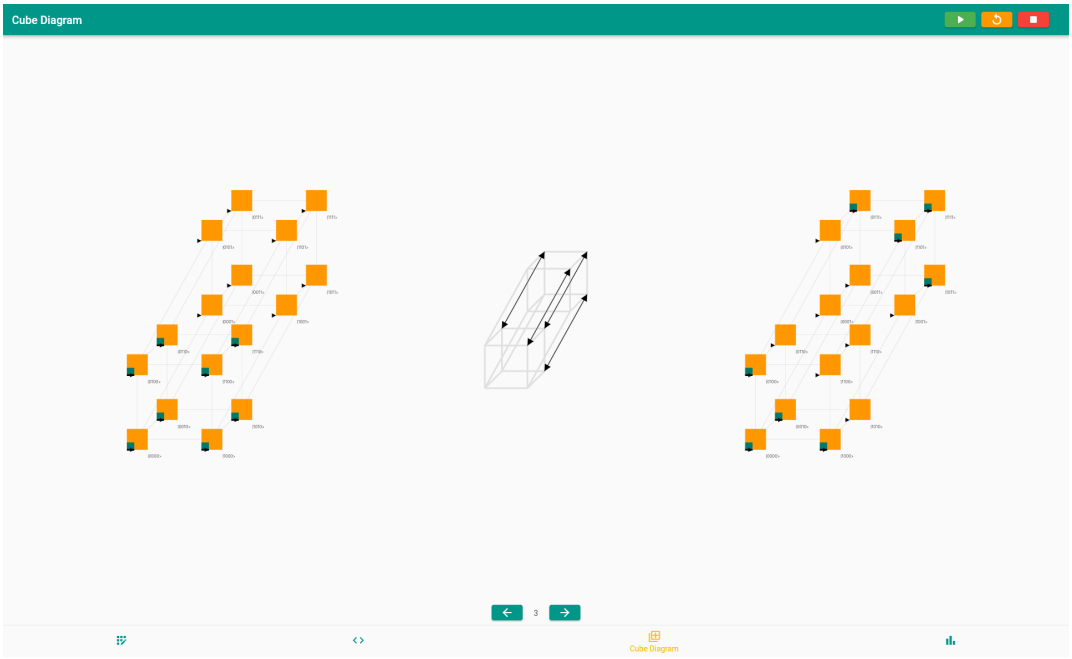


Abbildung 6.4.: Register vor und nach einem Orakel, sowie den Übergang

Im Plugin für die Visualisierung werden die *StateTransition* Instanzen durch den entsprechenden Painter in einem Canvas dargestellt. Abbildung 6.4 zeigt, wie der Zustand vor und nach einem Quantenorakel dargestellt wird, sowie den Übergang durch das Quantenorakel.

6.3. Simulation von „Controlled-U“

In Abschnitt 5.2 ist beschrieben, wie die Matrix aufgebaut ist, welche ein Kontroll-QuBit an eine bestehende Matrix anfügt. Für den Simulator wird die Matrix U an den Konstruktor der *CU* Klasse übergeben. Listing 6.5 zeigt, wie mithilfe von Controlled-U ein Controlled-Z Gatter aufgebaut und in einem Schaltkreis verwendet werden kann.

Eine Einschränkung bei der Implementierung ist, dass ein Controlled-U Gatter nicht auf beliebige QuBits angewendet werden kann. Nur die Konstellation aus Abbildung 6.5 ist möglich. Das Input QuBit muss das erste beteiligte QuBit sein, und die Output QuBits müssen nach dem Input QuBit folgen, ohne das unbeteiligte QuBits dazwischen liegen.

Listing 6.5: Controlled-Z Gatter in einem Schaltkreis verwenden

```
import 'package:quantumsimulator/quantumsimulator.dart';
var circuit = Circuit(
  initial: initial, //Initialer Vektor
  steps: [
    CU(0, Matrix.ofDouble([
        [1, 0],
        [0, -1],
    ])),
  ],
);
```

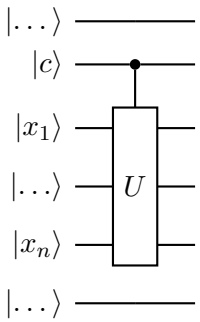


Abbildung 6.5.: Schaltkreis mit Quantenorakel

Eine Verallgemeinerung der Implementierung ist in Zukunft denkbar, überschreitet jedoch den Umfang dieser Arbeit. Listing 6.6 zeigt einen Ausschnitt aus dem Konstruktor und den Algorithmus zum Aufbauen der Matrix.

Listing 6.6: Aufbauen der Matrix für Controlled-U

```
// ID Matrix same size as matrix u, as List<List<ComplexDouble>>
var topLeft = Matrix.id(u.numColumns).elements;
// List<List<ComplexDouble>> of 0+0i same size as matrix u
var topRight = List.generate(
  u.numRows,
  (_) => List.generate(u.numColumns, (_) => ComplexDouble()),
);
// List<List<ComplexDouble>> of 0+0i same size as matrix u
var bottomLeft = List.generate(
  u.numColumns,
  (_) => List.generate(u.numRows, (_) => ComplexDouble()),
);
List<List<ComplexDouble>> res = [];
for (var i = 0; i < topLeft.length; i++)
  res.add(topLeft[i] + topRight[i]);// concat the top lists
for (var i = 0; i < bottomLeft.length; i++)
  res.add(bottomLeft[i] + u.elements[i]);// concat the bottom lists
return Matrix(res);
```

In Abbildung 6.6 ist beispielhaft zu sehen, dass zwar die Vektoren vor und nach der Berechnung von Controlled-U korrekt dargestellt werden können, für den Übergang bislang aber keine Visualisierung bereitsteht.

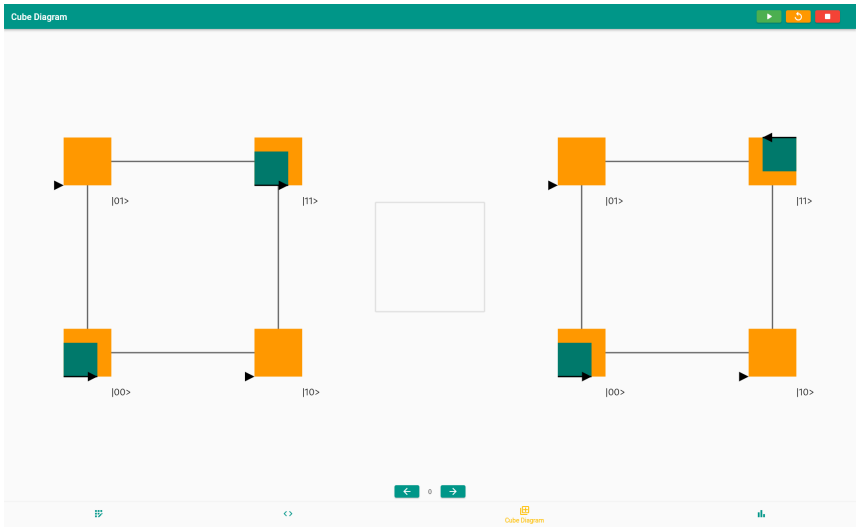


Abbildung 6.6.: Darstellung von Controlled-U

7. Fazit

Ein QuBit hat die Form $\alpha |0\rangle + \beta |1\rangle$. Wenn α und β beide ungleich 0 sind, spricht man von einem überlagerten Zustand oder von Superposition. Mehrere QuBits lassen sich als Register betrachten, kann ein Register nicht mehr durch Ausklammern in einzelne QuBits zerlegt werden, ist das Register verschränkt.

Ein Schaltkreis besteht aus einem Register auf dem Quantengatter angewendet werden. Jedes Quantengatter kann als eine Matrix dargestellt werden, welche die Auswirkungen des Quantengatters auf das Register beschreibt. Ein Messvorgang ist kein Quantengatter, das Messen eines überlagerten Zustands hat mit Wahrscheinlichkeit des Quadrats der Amplitude das jeweilige Ergebnis.

Registerzustände aus wenigen QuBits lassen sich als Würfeldiagramm visualisieren. Die Auswirkungen von Quantengattern und Messung lassen sich daran darstellen.

Ein Simulator der einen Quantencomputer widerspiegelt, braucht für die Berechnung komplexe Zahlen, sowie einige grundlegende Funktionen der linearen Algebra. Mit dem in Abschnitt 4.1 beschriebenen Softwaredesign lassen sich Schaltkreise zusammen setzen und diese mit beliebigen Registerzuständen initialisieren.

Phase Kickback ist ein Phänomen, welches in verschiedenen Szenarien auftritt. Die ersten beiden Szenarien verwenden dafür ein Quantenorakel zu einer unbekannten Funktion, im dritten Szenario wird ein Controlled-U Gatter verwendet. Mit der Implementierung der Anwendung kann gezeigt werden, dass auch Algorithmen die Phase Kickback verwenden simuliert werden können.

Die Anwendung ermöglicht es Schaltkreise aus beliebigen Quantengattern zusammenzusetzen. Eine Initialisierung des Schaltkreises mit einem komplexen Registerzustand ist ebenfalls durch das Softwaredesign sichergestellt. Damit erfüllt die Anwendung die ersten beiden funktionalen Anforderungen. Die dritte Anforderung konnte mit einem Plugin umgesetzt werden, das Plugin stellt jedoch keinen Support für die Web-Plattform zur Verfügung. Die Visualisierung ist als eigenständiges Plugin entwickelt worden und stellt Funktionen für die Darstellung von Registerzuständen, sowie Zustandsübergängen für die grundlegenden Gatter Hadamard, X, Y, Z, CNOT, Toffoli und Quantenorakel bereit. Die Visualisierung ist jeweils für 1 bis 4 QuBits implementiert und erfüllt damit die vierte Must-have-Anforderung.

Sowohl das Plugin für die Simulation, als auch das Plugin für die Visualisierung sind vollständig dokumentiert, zusammen mit den Tests die jede öffentliche Funktion auf Korrektheit überprüfen, erfüllt die Implementierung auch die nichtfunktionalen Anforderungen aus Abschnitt 3.3. Zusätzlich zu den automatisierten Tests wurde die

Anwendung ausgiebig auf den Plattformen: Windows, Linux und Android getestet. Hierfür wurde eine Auswahl an vorkonfigurierten Schaltkreisen angelegt, welche in der Anwendung simuliert werden können.

Im Anschluss an diese Arbeit soll weiter an der Anwendung entwickelt werden. Das wichtigste Feature, mit dem die Anwendung über die Proof of Concept Phase hinauswachsen wird, ist die Möglichkeit Quantenschaltkreise innerhalb der UI zusammenzubauen. Als Teil dieser Implementierung muss ein Parser für boolesche Funktionen entwickelt werden, um sicherzustellen, dass Quantenorakel auch aus der UI für beliebige Funktionen angelegt werden können. Daran anknüpfen kann die Auswertung der User Experience und die auf den Ergebnissen aufbauende UI/UX Design Verbesserung umgesetzt werden.

Ein Feature, welches schon zu Beginn einen positiven Einfluss auf die User Experience haben kann, ist eine interaktive Einweisung in die Anwendung, in Form eines In-App-Tutorials.

Bevor die Darstellung eines Übergangs durch ein Controlled-U Gatter implementiert werden kann, muss zunächst genau definiert werden wie diese Übergänge aussehen sollen. Auch für zusätzliche Gatter wie RX, RY und RZ muss zunächst eine geeignete Darstellung gefunden werden. Zusätzlich ist eine Verallgemeinerung der Implementierung von Controlled-U und Quantenorakeln denkbar, bei der die QuBits in beliebiger Reihenfolge verwendet werden können.

Weitere denkbare Features sind, die Entwicklung von Simulatoren, welche verschiedene Fehler einbauen, um daran Quantenfehlerkorrektur zu zeigen. Auch ist ein Simulator für „adiabatische Quanten Annihilation“ denkbar, um auch andere Modelle des Quantencomputing aufzuzeigen.

Die Entwicklung ist somit noch nicht abgeschlossen und eine Weiterentwicklung in den nächsten Semestern ist notwendig, um dass volle Potenzial der Anwendung auszuschöpfen.

Literatur

Bhat, Anvith. *Multithreading in Flutter Using Dart Isolates*. LogRocket Blog. 22. März 2022. URL: https://blog.logrocket.com/multithreading-flutter-using-dart-isolates/ (besucht am 27. 07. 2023).

Bloc State Management Library. URL: https://bloclibrary.dev/ (besucht am 23. 07. 2023).

BMBF. *Projekte - Quantentechnologien*. URL: https://www.quantentechnologien.de/forschung/projekte.html (besucht am 17. 08. 2023).

Dart Doc. URL: https://dart.dev/tools/dart-doc.html (besucht am 07. 09. 2023).

Einstein, A., B. Podolsky und N. Rosen. „Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?“ In: *Physical Review* 47.10 (15. Mai 1935), S. 777–780. DOI: 10.1103/PhysRev.47.777. URL: https://link.aps.org/doi/10.1103/PhysRev.47.777 (besucht am 12. 07. 2023).

Feynman, Richard P. *Quantenelektrodynamik: eine Vorlesungsmitschrift*. 2. Aufl. 1. Aufl. im Verl. Bibliogr. Institut Mannheim. München: Oldenbourg, 1989. 208 S. ISBN: 978-3-486-21256-3.

Flutter Team. *Flutter - Build Apps for Any Screen*. URL: //flutter.dev/ (besucht am 27. 06. 2023).

— *Testing Flutter Apps*. URL: https://docs.flutter.dev/testing/overview (besucht am 26. 08. 2023).

Hinton, C. Howard. *The Fourth Dimension*. Health Research Books, Feb. 1993. 286 S. ISBN: 978-0-7873-0410-2. Google Books: _ZG3MA1wvjIC.

IBM. *Qiskit*. URL: https://qiskit.org (besucht am 16. 08. 2023).

Just, Bettina. *Kursmaterial Woche 1 | Einführung in Das Quantencomputing - Teil 3 |*. URL: https://open.hpi.de/courses/qc-intro-3-2023/items/59gKFN1pHQufCdunvgr8AL (besucht am 29. 05. 2023).

— *Quantencomputing kompakt: spukhafte Fernwirkung und Teleportation endlich verständlich*. IT kompakt. Enthält: 1 Online-Ressource. Berlin Heidelberg: Springer Vieweg, 2020. 112 S. ISBN: 978-3-662-61888-2.

open HPI. *Quantum Computing*. URL: https://open.hpi.de/channels/quantum (besucht am 21. 08. 2023).

QCVIS. Fraunhofer-Institutes für Graphische Datenverarbeitung IDG. URL: https://fh-igd-iet.github.io/qcvis/ (besucht am 27. 07. 2023).

Qiskit.Visualization.Plot_bloch_multivector — Qiskit 0.43.2 Documentation. URL: https://qiskit.org/documentation/stubs/qiskit.visualization.plot_bloch_multivector.html (besucht am 12. 07. 2023).

QuanTUK. *DCN_Webtool*. URL: https://github.com/QuanTUK/DCN_Webtool (besucht am 16. 08. 2023).

Quirk: Quantum Circuit Simulator. URL: https://algassert.com/quirk (besucht am 17. 08. 2023).

Riverpod. URL: https://riverpod.dev/de/ (besucht am 07. 09. 2023).

Schrödinger, E. „Die gegenwärtige Situation in der Quantenmechanik“. In: *Naturwissenschaften* 23.48 (1. Nov. 1935), S. 807–812. ISSN: 1432-1904. DOI: 10.1007/BF01491891. URL: https://doi.org/10.1007/BF01491891 (besucht am 23. 06. 2023).

Senno, Gabriel, Thomas Strohm und Antonio Acín. *Quantifying the Intrinsic Randomness of Quantum Measurements*. Comment: 5+6 pages, 2 figures. Comments welcome! 7. Nov. 2022. DOI: 10.48550/arXiv.2211.03581. arXiv: 2211.03581 [quant-ph]. URL: http://arxiv.org/abs/2211.03581 (besucht am 14. 07. 2023). preprint.

The C4 Model for Visualising Software Architecture. URL: https://c4model.com/ (besucht am 06. 07. 2023).

Yin, Juan et al. „Bounding the Speed of ‘spooky Action at a Distance’“. In: *Physical Review Letters* 110.26 (26. Juni 2013). Comment: 11 pages, 4 figures to old, S. 260407. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.110.260407. arXiv: 1303.0614 [quant-ph]. URL: http://arxiv.org/abs/1303.0614 (besucht am 12. 07. 2023).

Glossar

Amplitude Komplexe Zahl, ein Vorfaktor eines Basiszustands.

Ana-Kata-Achse Richtungsangaben in der vierten Dimension (rechts, links, oben, unten, vorne, hinten, ana, kata).

Basiszustand Vektor eines Linear unabhängigen Vektorraums.

Betrag Nicht negative reelle Zahl, die den Abstand zu 0 angibt.

Bit Kleinste Dateneinheit in der Informatik, das nur einen von zwei Werten annehmen kann, welche üblicherweise mit 0 und 1 bzw. true und false bezeichnet werden.

BLoC Design Pattern und Software Bibliothek zum Statemanagement.

Canvas Digitale Zeichenfläche, die es ermöglicht grundlegende geometrische Formen zu zeichen.

Dartdoc Werkzeug zum Generieren der Software Dokumentation.

EPR-Paar Bekanntes Beispiel eines Registerzustands der nicht durch ein Tensorprodukt von einzelnen QuBits beschrieben werden kann.

Flutter Framework zur Entwicklung Plattformunabhängier Anwendungen.

Gatter Teil eines Schaltkreises, Kombiniert einen oder mehrere Inputs zu einem oder mehreren Outputs.

Ket-Notation Notation die auf Paul Dirac zurückgeht, zur Beschreibung quantenmechanischer Zustände.

Lokalität Änderungen an Objekten können mit maximal Lichtgeschwindigkeit einfluss auf andere Objekte haben.

Messung Siehe Abschnitt 2.4.

Quantencomputer Eine Rechenmaschine, die sich quantenmechanischer Prozesse bedient, um Berechnungen durchzuführen.

Quantengatter Teil eines Quantenschaltkreises, operiert auf einem oder mehreren Qubits.

Quantenorakel Schaltkreis der die Input-QuBits unverändert lässt und auf das Output-QuBit die Wirkung einer booleschen Funktion hat.

QuBit Kleinste Einheit im Quantencomputing, wird beschrieben durch $\alpha_0 |0\rangle + \alpha_1 |1\rangle$.

Realismus Eigenschaften von Objekten lassen sich messen und bleiben durch den Messvorgang unverändert.

Statevektor Beschreibung eines Registerzustands als Vektor.

Superposition Eigenschaft eines Registerzustands.

Tensorprodukt Besondere Form der Multiplikation von Matrizen oder Vektoren beliebiger Form.

Tesserakt Bezeichnung eines Würfeläquivalents der vierten Dimension.

Union-Type Datentyp, der verschiedene Datentypen vereinigt, die sonst keine Gemeinsamkeiten aufweisen.

Verschränkung Eigenschaft eines Registerzustands.

Anhang

I. Quantengatter

$$U_{ID} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad U_H = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$U_X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad U_Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad U_Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$U_{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$U_{\text{Toffoli}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

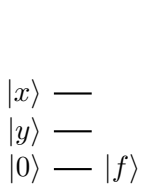
II. Zukünftige Features

- Schaltkreise in der Anwendung aufbauen.
- Speichern und Laden von Quantenschaltkreisen.
- Parser für boolesche Funktionen für Quantenorakel.
- Quantenorakel und Controlled-U auf beliebigen QuBits.
- Dimension von Würfel reduzieren, wenn gemessen wurde.
- Weitere Darstellungen für Statevektoren hinzufügen.
- Zusätzliche Gatter, wie zum Beispiel RX, RY, RZ und T.
- Darstellung für Controlled-U Gatter.
- Weiteres Simulator Backend für fehlerbehaftete Quantencomputer.
- Tutorial in der Anwendung
- UI/UX verbessern
- Adiabatic quantum annihilation.

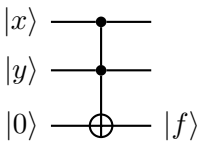
III. Boolesche Schaltkreise

	X	0	0	1	1	
	Y	0	1	0	1	Schaltkreis
0	0	0	0	0	0	Abbildung Ia
1	$X \wedge Y$	0	0	0	1	Abbildung Ib
2	$\neg(X \Rightarrow Y)$	0	0	1	0	Abbildung Ic
3	X	0	0	1	1	Abbildung Id
4	$\neg(Y \Rightarrow X)$	0	1	0	0	Abbildung Ie
5	Y	0	1	0	1	Abbildung If
6	$X \oplus Y$	0	1	1	0	Abbildung Ig
7	$X \vee Y$	0	1	1	1	Abbildung Ih
8	$\neg(X \vee Y)$	1	0	0	0	Abbildung Ii
9	$X \leftrightarrow Y$	1	0	0	1	Abbildung Ij
10	$\neg Y$	1	0	1	0	Abbildung Ik
11	$Y \Rightarrow X$	1	0	1	1	Abbildung Il
12	$\neg X$	1	1	0	0	Abbildung Im
13	$X \Rightarrow Y$	1	1	0	1	Abbildung In
14	$\neg(X \wedge Y)$	1	1	1	0	Abbildung Io
15	1	1	1	1	1	Abbildung Ip

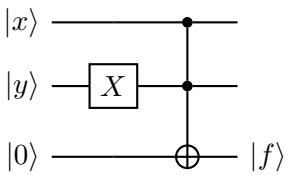
Tabelle 1.: Alle Funktionen der Form $f : \{0,1\}^2 \to \{0,1\}$



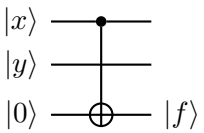
(a) $f(x,y) = 0$



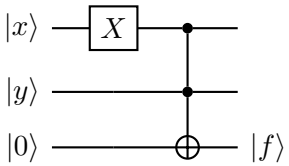
(b) $f(x,y) = x \wedge y$



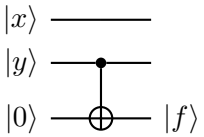
(c) $f(x,y) = \neg(x \Rightarrow y)$



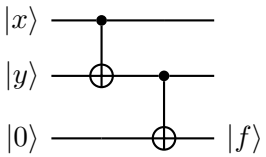
(d) $f(x,y) = x$



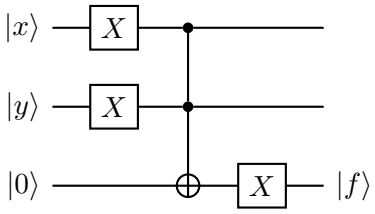
(e) $f(x,y) = \neg(y \Rightarrow x)$



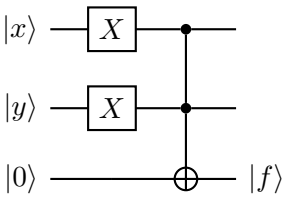
(f) $f(x,y) = y$



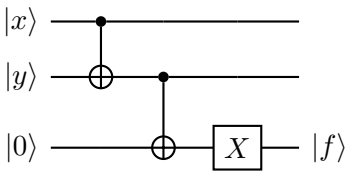
(g) $f(x,y) = x \oplus y$



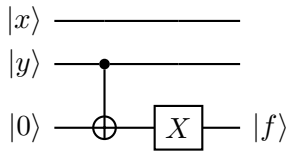
(h) $f(x,y) = x \vee y$



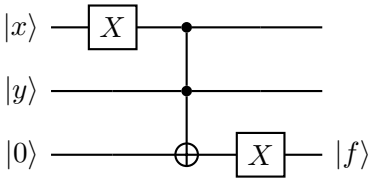
(i) $f(x,y) = \neg(x \vee y)$



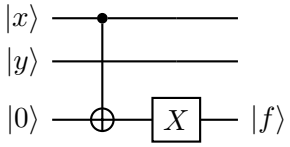
(j) $f(x,y) = x \leftrightarrow y$



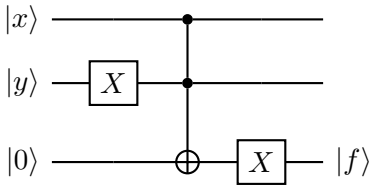
(k) $f(x,y) = \neg y$



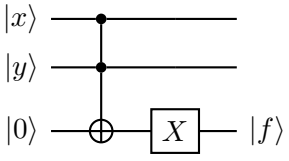
(l) $f(x,y) = y \Rightarrow x$



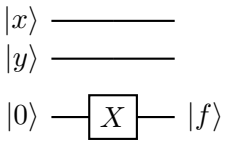
(m) $f(x,y) = x$



(n) $f(x,y) = x \Rightarrow y$



(o) $f(x,y) = x \wedge y$



(p) $f(x,y) = 0$

Abbildung I.: Schaltkreise zu booleschen Funktionen

IV. Liste der digitalen Anhänge

- Sourcecode der Anwendung
- Windows Build (.exe)
- Android Build (.apk)
- Linux Build
- Digitale Version dieses Dokuments